NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CATARINA AMARO GUILHERME DOS SANTOS CRISPIM

BSc in Electrical and Computer Engineering

SCALABLE DRL BASED ROUTING WITH TOPOLOGY CHANGES

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

March, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

SCALABLE DRL BASED ROUTING WITH TOPOLOGY CHANGES

CATARINA AMARO GUILHERME DOS SANTOS CRISPIM

BSc in Electrical and Computer Engineering

Adviser: Pedro Miguel Figueiredo Amaral
Assistant Professor, NOVA School of Science and Technology

Examination Committee

Chair: Rui Miguel Amaral Lopes
Assistant Professor, NOVA School of Science and Technology

Rapporteur: Luís Filipe Lourenço Bernardo
Associate Professor, NOVA School of Science and Technology

Adviser: Pedro Miguel Figueiredo Amaral
Assistant Professor, NOVA School of Science and Technology

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2024

Scalable DRL based routing with topology changes

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.18) [1].

To my friends and family.

Acknowledgements

Firstly, I would like to thank my adviser, Dr. Pedro Amaral, for the opportunity to develop this project and for his guidance and assistance.

I would also like to thank NOVA School of Science and Technology of NOVA University Lisbon and their teaching staff for all the knowledge provided during my degree.

I want to express my gratitude to the friends and colleagues who accompanied me during the last years.

My heartfelt thanks to Laura, my dearest friend, for always being there for me, for the insightful ideas and for all the moments of joy we shared.

I am deeply grateful to Miguel, for the endless support in more ways than one, for his inspiring positive outlook and for always uplifting me. This would have not been possible without him.

I would also like to thank my family, for their encouragement and support throughout this journey.

This work was funded by Instituto de Telecomunicações and Fundação para a Ciência e Tecnologia under the project UIDB/50008/2020.

Abstract

Optimizing network resources is a challenge, especially nowadays, where vast amounts of data are handled and there is a demand for a dynamic network capable of servicing different types of devices in the same infrastructure. Routing in particular is very hard to optimize, the routing problem is hard to model and its optimal resolution is of intractable complexity. Furthermore, the network dynamism makes the use of the model based solution impractical.

Deep Reinforcement Learning (DRL) is a suitable Artificial Intelligence (AI) technique for solving control problems without the need of a system model, and it has been proposed in the literature as a possible solution to solve the routing problem in networks. However, using DRL for routing has its own challenges. DRL agents have to be trained before being deployed and might not perform well when changes, such as link failures, occur in the network. There is also the dimensionality problem associated with the state and action spaces, which can be an issue for centralized approaches in many networks.

In this work, a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) DRL architecture is proposed to solve the routing problem. Both central critic and distributed critic options are considered using simple Q network and dueling Q network critic implementations. All combinations were trained in three different network topologies. To test the algorithms and evaluate their adaptability to small changes in the network, the original topologies used during the agent training process were changed, either with link failures or the addition on new ones, and a technique of continuous learning was employed.

It was observed that the dueling Q network is the better critic algorithm and that centralized critics perform better than local critics, for the majority of the scenarios. Regarding the adaptability to network changes, the algorithms adapted to varying degrees. Moreover, the usage of continuous learning further improved those results. Simulations in three different types of topologies show that the network topology is an important factor to consider when selecting the best algorithm combination for reducing performance loss in the presence of network changes.

Keywords: Multi agent, Deep Reinforcement Learning, Network routing

Resumo

No cenário tecnológico atual, otimizar recursos em redes é um desafio, uma vez que há uma elevada quantidade de dados a ser tratada e há uma necessidade de ter uma rede de encaminhamento dinâmica capaz de servir diferentes tipos de dispositivos na mesma infraestrutura. O encaminhamento é difícil de otimizar, o problema é difícil de modelar e a sua resolução ótima tem uma grande complexidade. Além disso, o dinamismo da rede torna o uso de uma solução baseada num modelo impraticável.

Deep Reinforcement Learning DRL é uma técnica de *Artificial Intelligence* AI adequada para resolver problemas de controlo sem necessidade de ter um modelo do sistema, e tem sido proposta na literatura como uma solução para resolver o problema de routing em redes. No entanto, usar DRL para routing tem os seus desafios. Os agentes de DRL têm de ser treinados antes de serem utilizados e podem não ter o comportamento esperado quando ocorrem alterações na rede, tal como falhas em caminhos. Há também o problema da dimensionalidade que está associado com os espaços de estados e ações, o que pode ser um problema para soluções centralizadas.

Neste trabalho, é proposta uma arquitetura de DRL com MADDPG para resolver o problema de encaminhamento. Foram utilizadas abordagens de *critic* central e local, e arquiteturas de *simple Q network* e *dueling Q network*. Todas as combinações foram treinadas em três tipos diferentes topologias de rede. Para testar os algoritmos e avaliar a sua adaptação a pequenas alterações na rede, as topologias originais de treino foram modificadas, com falhas em caminhos ou a adição de caminhos novos. Foi também utilizada uma técnica de aprendizagem contínua.

As implementações com *dueling Q network* obtiveram melhor desempenho, e os *critic* centrais são melhores que os locais na maioria dos cenários. Relativamente às mudanças na rede, os algoritmos adaptaram-se de forma diferente. A utilização de aprendizagem contínua melhorou os resultados. As simulações feitas nos três tipos diferentes de topologias mostram que a topologia de rede é um fator importante a considerar ao escolher o algoritmo para reduzir a perda de desempenho quando ocorrem alterações na rede.

Palavras-chave: Multi agente, *Deep Reinforcement Learning*, Encaminhamento

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence *(pp. iv, v, 5)*

ARPANET Advanced Research Projects Agency Network *(pp. 14, 33, 35, 39, 42, 44, 51, 53, 54, 57)*

AS Autonomous System *(pp. 5, 18, 34)*

BGP Border Gateway Protocol *(pp. 5, 18)*

DDPG Deep Deterministic Policy Gradient *(pp. 12, 14, 15, 18–21)*

DL Deep Learning *(pp. 7–9)*

DNN Deep Neural Network *(pp. 7, 8, 10, 17)*

DPG Deterministic Policy Gradient *(p. 11)*

DQN Deep Q-Network *(pp. 9, 10, 12)*

DRL Deep Reinforcement Learning *(pp. iv, v, 1, 2, 8, 12–18, 21, 28, 31, 33, 38, 48, 50, 54, 57–59)*

DROM DDPG Routing Optimization Mechanism *(pp. 15, 16)*

ECMP Equal-Cost Multi-Path *(p. 15)*

GNN Graph Neural Network *(pp. 16, 17, 59)*

GRU Gated Recurrent Unit *(p. 15)*

MADDPG Multi-Agent Deep Deterministic Policy Gradient *(pp. iv, v, 19, 21, 31, 58)*

MARL Multi-Agent Reinforcement Learning *(pp. 8, 18)*

MDP Markov Decision Process *(pp. 7, 19, 21, 22)*

ML Machine Learning *(pp. 5, 7, 8, 12, 31)*

NN Neural Network *(pp. 6, 7, 9, 18, 31, 45, 58, 59)*

NSFNET National Science Foundation Network *(p. 14)*

OSPF Open Shortest Path First *(p. 16)*

PG Policy Gradient *(pp. 11, 12)*

ReLu Rectified Linear Unit *(pp. 26–28)*
RL Reinforcement Learning *(pp. 7, 8, 31)*

SDN Software Defined Network *(p. 15)*
SPF Shortest Path First *(p. 15)*
SPG Stochastic Policy Gradient *(p. 11)*

TD Temporal Difference *(p. 9)*
TE Traffic Engineering *(pp. 1, 13–15, 18, 19)*
TL Transfer Learning *(pp. 12, 13, 59)*

1

Introduction

Traditional routing algorithms, such as Dijkstra, that operate on choosing the shortest path between nodes are known to have limitations in achieving an optimal utilization of network resources in complex and dynamic networks. For instance, one of the problems with these algorithms is that they do not take the whole network state into consideration when calculating the routes, making them less efficient when dealing with intricate network topologies where they are unable to achieve optimal use of network resources [2].

Classic solutions, based on optimizing the link weights in shortest path algorithms, an approach that is suitable for simpler networks, have been proven to be NP-complete (nondeterministic polynomial-time complete), meaning that the bandwidth optimisation problem cannot be solved in an algorithmic way, only by trial and error [3]. As a result, applying a solution of this nature to a complex network is not practical. In fact, modeling traffic routing as an optimization problem is, in general a hard problem. Modern networks are challenging to model considering their dynamic nature and the wide variety of services they are required to support. Therefore obtaining a mathematical model that accurately represents a real dynamic network routing system is very hard to do. Hence the demand for a more flexible approach capable of autonomously computing the best solution without the need of a system model and efficiently performing Traffic Engineering (TE).

Recently, model-free solutions using DRL techniques have been proposed to manage routing in modern networks. DRL can be used for autonomous learning of network processes and effective decision making in dynamic systems. However, these approaches face two main challenges:

1. Difficulty to converge due to large action spaces when networks scale up, something usually referred to in DRL models as the curse of dimensionality [4]. This happens because of the vast amount of actions to choose from in the pursuit of an optimal policy function.

2. Low resiliency to changes as a consequence of training in a specific network topology, and consequently not knowing how to behave in other topologies.

1.1 Motivation

Routing algorithms based on DRL can be categorized into two main sub-groups, single agent and multi-agent.

1. Single agent solutions can be described as a centralized approach. When working with an extensive network, this approach has a scalability limitation because the agent state and action spaces can grow exponentially. As a result, during the training process the agent will become overloaded and the algorithm will converge slowly.

2. Multi-agent approaches are decentralized and consequently do not allow for a global view of the network and the various nodes that are in it. Although it reduces the amount of information each agent has, this approach also reduces the processing load each agent has to handle since the network information is split between various agents. This way, a complex problem can be broken down into smaller portions, making it possible for DRL algorithms to be applied to an elaborate network scenario.

In this dissertation we consider the multi-agent approach to scale the use of DRL based routing and aim to study the trade-offs between achieving agent scalability, by locally training several agents, and the achievable routing optimally, using independent learning without global knowledge.

To further evaluate the DRL algorithms for routing networks, the impact of topology changes in performance will be studied. We aim to investigate whether rapidly changing the action space, when changes are made in the original training topology, improves performance even when the new topology configuration was not used during the training of the agents, despite the agents being trained with a different action space, even though the dimensions stay the same.

1.2 Objectives

The aim of this dissertation is to maximise link bandwidth in a routing network. Various multi-agent algorithms will be applied to a dynamic routing scenario with the intent of comparing their behaviours in different topologies. A second goal is to study the impact of topology changes in performance and observe if continuously training the agents in the new scenario can improve results. The routing algorithms use scalable DRL techniques with distributed agents.

1.3 Outline

This document is divided into six chapters.

Chapter 1 provides an overview of the scope, motivation and goals of this dissertation.

Chapter 2 introduces a theoretical foundation in the context of this project, as well as relevant research papers.

Chapter 3 describes the methodology used in the development of the project.

Chapter 4 describes how those techniques were implemented, and specifies the experiences that were performed.

Chapter 5 presents the results of the experiences and their discussion.

Chapter 6 summarizes the findings and identifies limitations and future work.

2

State of the art

2.1 Routing

Routing can be performed in various ways. There are some routing solutions that focus only on transmitting information without delay times and quality of service in mind, while others take those aspects into consideration. As mentioned previously, traditional routing solutions have optimization issues and therefore are not capable of performing optimally in complex networks.

Shortest path algorithms, where traffic flows to the destination through the lowest cost paths while not considering eventual congestion and delay, are the basis of routing algorithms. They calculate the best possible route, for the metric considered and from the information of the full network topology. Different metrics can be used such as number of hops it takes to reach the destination or the physical distance between the start point and the destination.

The limitation of shortest path algorithms is that they are static, and thus only take into account the routes established in the discovery process. Based on the shortest path logic, there are two types of dynamic algorithms: distance vector and link state. The distance vector routing method uses a table for each router that stores the best known distance to every destination and the link to use in the next hop to get there. To reach a position where every router knows the best possible routes, these tables are exchanged by the router with their neighbouring peers to compare and update the best distances and links. This algorithm has a critical flaw when the topology changes that can cause it to converge very slowly. The link state routing method, that addresses the distance vector's slow convergence, works by each router forwarding packets that contain information about its links. After receiving the same information from other routers, it has a view of the topology and then computes the paths to every router.

Link state protocols face scalability challenges in very large networks, and although they converge faster than distance vector protocols, they still do not provide optimal routing [5].

Both distance vector and link state shortest path routing protocols are widely used

across different network levels, for their simplicity and robustness.

In a network comprised of multiple areas, each different area can be considered as its own network and is known as an Autonomous System (AS). When performing routing, it must be taken into account if it is intra-domain, inside an AS, or inter-domain, between two ASs. In intra-domain routing, link state routing is preferred due to being more suitable for smaller network sizes and having higher convergence speed. However, in inter-domain routing a variant of distance vector protocols, named path vector, is used. Path vector protocols are more effective for larger networks because of better scalability. For optimization, for intra-domain link state protocols, it entails of adjusting link weights, whereas for inter-domain path vector it entails of adjusting route attributes [5].

Considering Border Gateway Protocol (BGP), an inter-domain protocol that performs routing in the internet between ASs, the framework of this protocol is that independent clusters of nodes are connected and transfer data between them [5]. However, this approach has some limitations, such as, information shared between regions being limited. In a similar system, with a comparable degree of complexity, control over such a large area can not be achieved in a simple manner with only one agent overseeing the whole network traffic and managing congestion, because it would not be possible to consider every possible option and determine the best one.

Applying AI and Machine Learning (ML) techniques to routing algorithms is a possible solution to assist in optimizing networks that can not be represented by a system model.

2.2 Machine Learning

ML is the capacity that a system has of observing and learning patterns based on certain indicators called features. Features are specific attributes of a data set and are pre-selected either with the assistance of an algorithm or through manual input. As a result, the algorithm then knows to focus on those portions of the studied data.

It works by analysing the information, separated in a training and learning set. The training set is used to build the model. The ML algorithm learns from this data by observing how those established indicators change through the various iterations, while training. It is then possible for the system to identify patterns and make inferences, such as relationships in the data. The validation set, used when working with labeled data, helps improve the final model. This batch of data is used to tweak the learning algorithm's parameters for a more accurate representation [6] [7].

There are various learning methods for agent training, the main ones being unsupervised, supervised and semi-supervised learning. These methods differ from each other by the degrees of labels in their input datasets [6] [8].

1. Unsupervised learning recognizes patterns in the input data and deciphers its connections. In this approach, the input data is not labeled and it is up to the algorithm to find correlations between datasets.

2. Supervised learning is used when an agent is being trained to respond to new conditions based on real data that it had as input. This data is labeled and the algorithm has to, based on this knowledge, figure out an approximation of how a system will behave when faced with unknown conditions.

3. Semi-supervised learning, that is in the realm of the hybrid learning styles, combines labeled and unlabeled data.

4. Reinforcement Learning, a method based on acquiring knowledge from experience and results. It is well suited for control problems such as routing and therefore it will be covered in more detail in this chapter.

2.2.1 Neural Networks

Artificial Neural Networks, or Neural Networks (NNs), are systems that process data through a structure that emulates the way neurons propagate signals in the human brain. In the NNs' architecture, a neuron is equivalent to a node, and the way electric impulses transmit information corresponds to the algorithm that processes the input data. These networks can analyse and establish links between data and perform operations such as regressions, clustering and data forecasting [9].

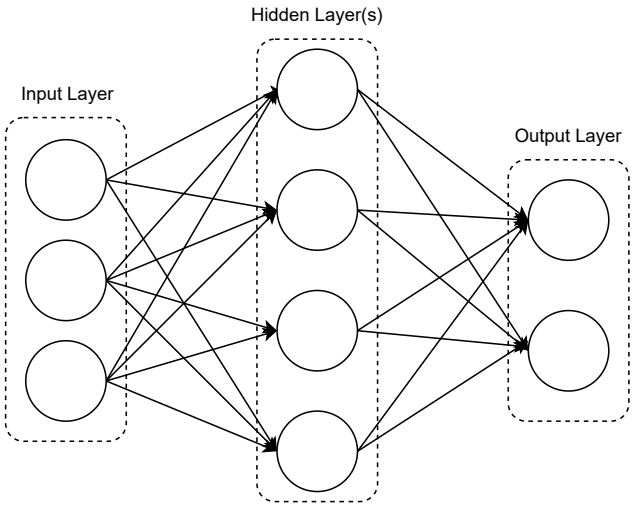

Figure 2.1: Neural Networks framework diagram (adapted from [10])

Figure 2.1 illustrates the framework of one kind of NN which is composed by:

1. Input layer, that receives the data

2. Hidden layer, or layers, depending on the application context, are intermediate layers that entail the processing segment

3. Output layer, that displays the final result

Each node has connections that have weights and thresholds associated with them. The weight of a connection differentiates the significance of a determined variable in the desired outcome of the experience. If a parameter is substantial enough, and is above the threshold, a function will be activated.

2.2.2 Deep Learning

Deep Learning (DL) differs from ML in the sense that it can process data with greater levels of abstraction. This approach is useful because it allows the use of more extensive data sets without needing the intervention of selecting and organizing features.

DL uses NNs with multiple layers, Deep Neural Networks (DNNs), to process data more thoroughly, as each layer works on the processed data from the layer before, and thus achieving a higher level data analysis [11].

2.2.3 Reinforcement Learning

Reinforcement Learning (RL) extracts knowledge from the result of trial and error experiments and using a positive feedback loop approach. Each action performed has an outcome that will impact the system in some way, after assessing if that impact was positive or negative in relation to the end goal, a policy function will be positively or negatively rewarded. The systems constantly analyses actions and their effect [12].

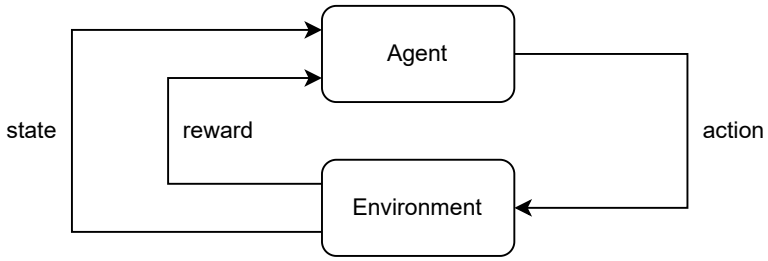

Figure 2.2: Reinforcement Learning behaviour diagram (adapted from [12])

The figure 2.2 illustrates the conceptual structure that is behind the RL behaviour. The agent being trained has a set of actions in its action space, which represent all the possible acts the agent can execute. The environment is represented by a state and a reward, based on the impact a determined action had on the system and if that impact directed the system closer to the end goal or not. The agent first examines the state of the environment to compute which action will be more suitable next based on the policy function. This function holds the information for the best action to take for a certain state. Following that action, there will be a consequence in the environment and the reward function will be updated. This function defines the potential rewards associated with a determined action for a specific state. The goal of this process is to find a policy that maximizes the reward. This method can be modeled as a Markov Decision Process (MDP), a stochastic process, that mathematically represents the probability of a state change. This model represents a

probability associated which each state transition considering the environment state and the action [13] [14].

2.2.3.1 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a decentralized method for controlling networks, where each agent can be instructed to supervise different activities. Each agent has each own learning process and performs its own tasks, but cooperation between the agents can be established to increase the understanding of the environment they are in and what actions to take to achieve the end goal.

2.2.4 Deep Reinforcement Learning

In ML, the term DRL refers to the integration of DL with RL. Combining these two techniques makes the use of RL possible for problems involving a higher number of state-action pairs, since the agent no longer needs to memorize outcomes in table form, instead a DNN is used to learn that data.

The algorithms for implementing DRL can be classified into model-based and model-free. The model denotes the environmental data that is known. In a model-based method, the algorithm can use an environment model to infer rewards and direct the exploration of actions, which can accelerate convergence. In a model-free method, as the environment model is unknown, the algorithm needs to explore the environment and learn the reward function from experiments without the assistance of a model.

In the context of this thesis, the focus is on model-free approaches, given the complexity of modern networks [12].

Model-free DRL optimization algorithms can be split into three categories: value-based, policy-based and actor-critic, a method that results from combining the first two.

These methods can also be split into two other categories based on how the policy function is improved: on-policy or off-policy. On-policy methods only rely on the interactions the agent has with the environment to optimise the policy function, while off-policy methods do not. The latter can use previously gathered information to improve a separate policy function, distinct from the one used to interact with the environment in that episode [12].

This is also known as exploration and exploitation. Exploration refers to the off-policy method, where in order to achieve the optimal policy function, all the alternative actions are considered and studied regarding the impact they have in the system and how that translates to the end goal. So, the actions chosen with these might not be the most appropriate considering the agent knowledge, but they are still chosen and evaluated. Exploitation refers to the on-policy method, where the agent keeps choosing actions that most align with the optimal result, taking the risk of overlooking potentially valid options. The balance between exploration and exploitation is an important aspect of DRL algorithms, an example of a simple method to try and achieve it is the ϵ-greedy policy:

- with ϵ being the probability of the agent performing an exploratory action
- with $1 - \epsilon$ being the probability of the agent performing an exploitative action

2.2.4.1 Value-based methods

Value-based methods focus on improving the value function for policy optimization [12]. The value function describes the cumulative reward for a determined state and represents the set of actions that will achieve it. This function can propose actions that might not bring a high reward instantly but will lead to a higher final reward [15].

During the learning phase, these methods offer a more thorough exploration component, considering these algorithms examine all possible options [16].

Value-based algorithms are based on Q-learning, an off-policy Temporal Difference (TD) method for optimizing the Q-function. For this reason, it focuses on improving the value function with information gathered from the whole experience. A TD algorithm uses the difference between states at distinct times in its learning process, and tries to predict a reward from that data. The Q-function organizes the state and the expected long-term reward value of each action in an action-value function, as a table. This multivariable function, represented in equation (2.1), has an input pair of the current state and action to take that equals to one output, the expected reward [12].

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.1}$$

In this equation:

- $Q(S_t, A_t)$ is the state-action pair
- α is the learning rate, which is the value that controls how often the state-action values are updated
- R_{t+1} is the reward associated with the transition from state S_t to state S_{t+1} after executing action A_t chosen by the policy
- γ is the discount factor, which sets the importance of that action considering if a reward is instantaneous or delayed
- $max Q(S_{t+1}, a)$ is the maximum reward considering every action, not taking policy into account, just the state-action function

To calculate the updated reward for a given state-action pair $Q(S_t, A_t)$, the current value of that pair is summed with the anticipated reward for that action. After each iteration of the algorithm the Q-function is updated and the process continues until convergence [13].

Deep Q-Network (DQN) results from the combination of Q-learning and DL, where the Q-function is approximated by a NN and provides the algorithm with properties to stabilize its learning such as the experience replay and target network techniques [17]. This

approach has a much bigger computational power than Q-learning, because DQN uses a DNN for data processing while Q-learning only uses the Q-table and thus it can work with high dimensional state spaces. Although being able to handle larger state spaces than Q-learning, this approach is not suitable for very large or continuous action spaces.

Considering the behaviour of the Q-learning algorithm, each iteration can lead to a different result, and a continuous disparity in reward results can lead to the divergence of the Q-function, making it unstable. The target network method is used for this issue, the initial policy function values are saved separately and are only updated to match the optimised policy periodically. It aims to increase the Q-learning algorithm reliability by reducing the frequency of Q-value updates, and thus diminishing Q-value overestimations, namely when the sate S_{t+1} is similar to state S_t. When two states are too similar, the agent could falsely boost the reward when in practice, that action would not immediately translate to a better environment [13] [18].

Another problem that can arise is the formation of unwanted correlations between data, this problem arises from data being considered consecutively. This issue can be solved with the experience replay technique, which randomizes the collected batches of data, used on the Q-function, to eliminate unwanted correlations. This technique consists of saving a set of agent's experiences, depicted by the tuple $e_t = (S_t, A_t, R_t, S_{t+1})$ in a replay buffer. Values from the replay buffer are then chosen at random, which significantly reduces the amount of interactions needed with the environment, improving sample efficiency and contributing to a lower variance of learning updates [13] [18].

The algorithm 1 shows the training loop for DQN [12].

Algorithm 1: DQN

Initialize: replay buffer capacity, discount factor, Q-network with weights θ, target $\hat{Q}$-network with weights $\hat{\theta}$, target network update rate

for *each episode* **do**
- Observe environment
- With ϵ probability choose random action A, or with $1 - \epsilon$ probability choose an action A from the state-action pair function that offers maximum reward
- Observe environment changes, state S_{t+1} and reward R_t and store them in the replay buffer (S_t, A_t, R_t, S_{t+1})
- Randomly pull various sets of (S_i, A_i, R_i, S_{i+1}) values from the replay buffer
- **if** *a tuple is from the end of an episode* **then**
  - Set the Q-learning target to $Y_i = R_i$
- **else**
  - Set the Q-learning target to $Y_i = R_i + \gamma max_{a'} \hat{Q}(\phi_i', a; \hat{\theta})$
- **end**
- Perform gradient descent algorithm on $(Y_i - Q(\phi_i, A_i; \theta))^2$
- Update the target network's weights with the primary Q-network weights at the target network update rate

end

2.2.4.2 Policy-based methods

Policy-based methods work on improving the policy function directly, based on defined parameters. The policy function is constantly improved according to the rewards until it can no longer improve and reaches convergence [12]. This function represents the choice of actions the agent has for a state in order to reach a desired outcome. This approach combats some issues associated with value-based based methods such as complications that result from states the agent is unable to comprehend or process and the degree of complexity when dealing with large state and action spaces.

Some of these methods are:

1. Policy Gradient (PG)
2. Stochastic Policy Gradient (SPG)
3. Deterministic Policy Gradient (DPG)

The policy optimization process can be classified into gradient-based, that use the policy gradient and gradient-free methods, a simpler method, because since it does not have to delve into the complex derivative gradient operations, therefore the learning process is faster.

Policy gradient methods use on-policy algorithms to train the agent. It adjusts the policy function to perform actions that achieve a higher cumulative reward until it reaches the maximum expected reward [12].

The equation (2.2) represents the PG theorem [12].

$$\nabla_\theta J(\pi_\theta) = E_{\tau \approx \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta(\log \pi_\theta(A_t|S_t))Q^{\pi_\theta}(S_t, A_t)\right] \tag{2.2}$$

In this equation:

- π_θ is the policy function
- θ are the policy parameters
- $J(\pi_\theta)$ is the goal of the agent
- $E_{\tau \approx \pi_\theta}$ is the expectation
- $\pi_\theta(A_t|S_t)$ is the policy function that represents the probability of choosing action A based on State S
- $Q^{\pi_\theta}(S_t, A_t)$ is the state-action function, based on π_θ

Based in the PG theorem, there are two methods to optimize policies that are either stochastic or deterministic. The SPG method, with a stochastic policy, expresses the action as a probabilistic distribution according to a given state. Whereas, the DPG method, that has a deterministic policy, represents a determined action as a solution to a particular state [12].

2.2.4.3 Actor-critic methods

The actor-critic methods combine the value-based Q-function for better sample analysis performance and policy-based techniques to develop a policy function. Each function is associated with an element, the actor for the policy function and the critic for the value function. Additionally, they cooperate in a way that the actor chooses an action based on the policy function and the critic analyses their performance in the environment [12].

The Deep Deterministic Policy Gradient (DDPG) method emerged from the adaptation of DQN with Deterministic Policy Gradients to successfully handle a continuous action space [19]. This approach uses the DQN architecture for the critic and a PG algorithm for the actor. This technique uses a target network strategy for both the actor and the critic, where the target values are slowly and exponentially incremented which differs from the one used in DQN where the values are just copied. Although this task slows the learning process, it is outweighed by the gain in learning stability of the action-value function [12]. Another strategy used is the addition of noise to the actor in order to control the exploration and exploitation equilibrium, by adding noise to the actor policy it is possible to steer the learning process to exploration. This exploration issue is common in scenarios with continuous action spaces.

One variation of this algorithm was proposed in Menghao et al. [20], where in the critic module, the Q value output is calculated with a dueling Q network. In the critic network, with the dueling Q network, the action and the sate are separated into value estimation and advantage function, to compare how good an action is for each state. This method proved to be more efficient that the regular actor-critic.

2.2.4.4 Comparison between methods

Table 2.1 summarizes the key points regarding each aspect of each type of DRL method and in relation to each other.

2.3 Transfer learning

Transfer Learning (TL) is a ML method that uses knowledge extracted from a previous task and applies it to a new similar task.

As illustrated in figure 2.3, this technique uses a trained model from a previous task and applies it to another task, with a new input dataset. These two tasks have to share specific factors between them, so that the information previously learned is relevant to the new context.

This method is useful because as the learning process is not a separate event, but an aggregate of experiences, the outcome is a versatile model that can be applied to various other environments. It allows for reduced computing time when, as the new task already has a knowledge foundation in some key aspects, it just had to employ it in the new circumstances.

Table 2.1: Comparison of Deep Reinforcement Learning methods key aspects

	Value-based	**Policy-based**	**Actor-critic**
Approach	Agent optimizes state-action function and chooses policy based on that	Agent optimizes policy function directly	Agent has an actor network that is in charge of the policy function and a critic network that learns the value function
Advantages	In the right context can be more sample efficient than policy-based; Low estimation variance	More straightforward policy parametrization. No resources needed for optimizing the values of the action space	Benefits of value and policy-based
Disadvantages	Comprehensive learning process which takes more time and can lead to overestimation	Slow convergence for global goal	Greater network architecture and training complexity
Convergence	Slower	Faster for maximizing the local goal, as opposed to the global goal	Combination of Value-based and Policy-based
Policy estimation	Exploration	Exploitation	Exploration and exploitation
Scalability	Does not perform well with a continuous action space	Can withstand broad and continuous action spaces	Suitable for a continuous action space

In the context of this thesis, when dealing with a new network topology, similar to a preceding one, using TL appears to be a promising approach to improve convergence times.

When TL is applied in a setting that does not share enough similarities with the knowledge model, a phenomenon called negative transfer can occur, which will hinder the learning process in the new environment [8].

2.4 Related Work

Xu et al. [2] proposed a model-free DRL control framework, DRL-TE, that encompasses two new strategies to improve TE in a network topology: TE-aware exploration and actor-critic-based prioritized experience replay with the aim of maximizing network utility. This

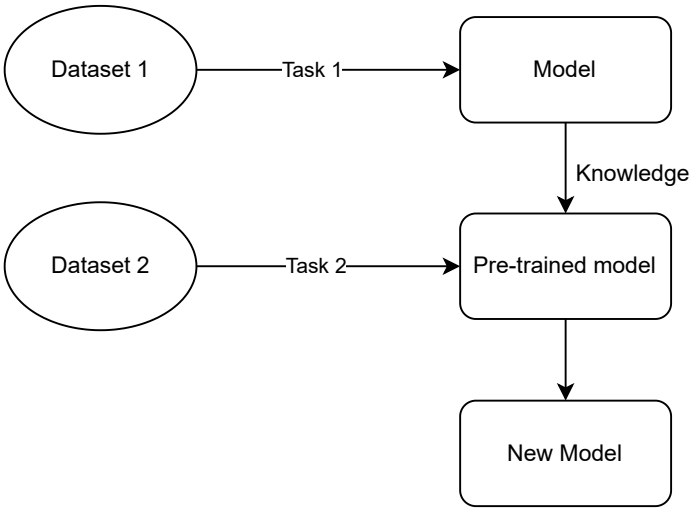

Figure 2.3: Transfer Learning process diagram (adapted from [8])

study defines the problem as a DRL problem with the following state, action and reward definitions:

- State space: is composed by the throughput and delay of each communication session
- Action space: is composed by the action vector of all the possible options to execute, such as the probability of choosing a path for a communication session, that is represented by a set of split ratios
- Reward: is the sum of utility of all communication sessions, the utility function considers the throughput and delay for a session

From the performed simulations, the authors found that DDPG did not perform well in their TE problem. They found two main causes: a lack of specificity when exploring the network and the use of a sampling method that overlooks the importance of transition samples in the replay buffer. The technique proposed as TE-aware exploration uses a random probability, ϵ, to decide which course to take. This probability value, ϵ, defines if the action to take will be based on the actor network or a base action with a random noise, N added:

- with $(1 - \epsilon)$ probability the agent performs $a + \epsilon \cdot N$, a is the output of the actor network $\pi(\cdot)$
- with ϵ probability the agent performs $a_{base} + \epsilon \cdot N$, a_{base} is a base TE solution

The value of ϵ decreases along the epoch progression to increase the accuracy of the actions performed. After performing experiments in two known topologies: Advanced Research Projects Agency Network (ARPANET) and National Science Foundation Network (NSFNET), and a random topology, it was concluded that DRL-TE improves network

performance. The solution showed reduced end-to-end delay, improved throughput and is resistant to network changes.

In Sun et al. [21] an approach based on DRL named ScaleDRL is proposed for TE in a network with dynamic traffic. The concept of the ScaleDRL algorithm is to pick a section of critical links, from the available network links, and dynamically change the weights on those links. This algorithm is based on pinning control theory, that states that it is possible to achieve efficient network control by only controlling a few elements as opposed to the entire system, which would consume too many resources, so in this case it translates to a smaller action space. Firstly, in the offline phase, the algorithm distinguishes critical links from ordinary links based on the centrality of each link. Secondly, in the online phase, the algorithm uses an actor-critic DRL algorithm to adjust the critical link weights to control network traffic. To the actor-critic configuration Gated Recurrent Unit (GRU) is added to extract temporal characteristics in the traffic data and after that a feedforward network. To endorse exploration of the DRL algorithm, a random value vector is added to the chosen action. ScaleDRL uses the following state, action and reward model:

- State space: traffic distribution on each link.
- Action space: array with the critical links and the corresponding possible routes.
- Reward: in this case is end-to-end delay, but varies considering the aim of the problem.

This approach is interesting since it concentrates on the critical nodes rather than attempting to manage the entire network, which provides better scaling of the action and state spaces. By comparison with other algorithms such as Shortest Path First (SPF), Equal-Cost Multi-Path (ECMP), TIDE and DRL-TE , ScaleDRL shows lowest end-to-end delay.

DDPG Routing Optimization Mechanism (DROM) is a DRL mechanism proposed by Yu et al. [22] to optimize routing in a Software Defined Network (SDN) with DDPG. The standout feature in this work is the ability to tune the importance that different settings (i.e. delay, throughput, etc) can have in the training process, and the action space link weights will change according to that. DROM agent framework:

- State space: is a traffic matrix that depicts the current network load,
- Action space: is an array of weights for the network links.
- Reward: can be evaluated by different metrics such as delay, throughput, hops or a combination of them.

Based on the observed environment, that is represented in the traffic matrix, the agent changes link weights in order to control how traffic flows. After the action is performed,

the new environment is analysed and the agent can determine if the outcome is favourable or not and calculate the reward. The results of performance comparison, with the goal of optimizing delay, between random routing configurations, Open Shortest Path First (OSPF) and DROM reveal that DROM has lower delays. To evaluate scalability, DROM is compared with two throughput optimization methods, QAR and SDN-LB, and the results show that DROM has an accentuated advantage as the traffic load increases.

In Sun et al. [23] a scalable routing strategy is proposed, ScaleDeep. This approach uses an algorithm to choose critical nodes on the network. These nodes are capable of replicating an accurate network performance. Similarly to using pinning control theory to choose critical links in Sun et al. [21], in this work, the authors choose critical nodes. This technique controls the entire network with only a few elements, rather than controlling them all. By choosing critical nodes, named driver nodes, instead of looking at the whole network, the algorithm focuses on the driver nodes and consequently its processing load is significantly reduced and scaling is more viable because the agent training does not rely on a set network architecture. Firstly, the driver nodes are selected in the offline phase. The distinction between driver and follower nodes is made by an algorithm that analyses the number of links connected to a node and determines which ones have greater importance in the network structure. Secondly, in the online training phase, the routing policy is improved based on the state of the network and the driver nodes. Based on the collected data, to improve network performance, a DRL algorithm manipulates the network traffic routes by dynamically modifying link weights after accessing the traffic flow on the driver nodes. Then, using a weighted shortest path algorithm based on the changed link weights, new forwarding paths are calculated. ScaleDeep uses the following state, action and reward model:

- State space: matrix with traffic throughput on driver nodes.
- Action space: array containing the link weights of the driver nodes connections
- Reward: average flow completion time, but can vary according to the metric considered.

From simulations performed, it is possible to see that ScaleDeep is robust to changes in the network when a follower node fails, but not when a driver node does.

The integration of Graph Neural Networks (GNNs) with DRL, designated as the DRL+GNN method, has been proven to be successful in providing a DRL based routing agent with the capacity to generalize unseen network topologies, thereby being able to operate in a network that was not used in its training process [24]. The approach proposed in Almasan et al. [24] proposes the use of GNNs to allow training an agent with one topology and being able to apply it in a different one, given that they have some characteristics in common. In the training process, the agent receives each network state observation

in a graph structure as input. Afterward the GNN takes that topology information and processes it as a graph model where the network links relationships are represented by graph nodes. After this output being processed by a DNN, the GNN produces a q-value estimate that will then be evaluated over a number of actions and ultimately, the action with the largest q-value will be chosen.

Despite these works referring to single agent approaches, these algorithms and concepts can be translated into a multi-agent learning scenarios

2.4.1 Multi-Agent methods

In Geng et al. [25], a multi-agent DRL approach is proposed to manage traffic in a multi-region network. Using multi-regions is an important factor to decentralize the decision making process of the whole network optimization, improve scalability and improve the ability to withstand link failures. Information synchronization when dealing with multi-regions can be difficult to manage, so this paper proposes that each area of the network should have two reinforcement learning agents. One that is only concerned with terminal traffic, going to that local network, the T-agent, and one that supervises outgoing traffic, that is inter-region, the O-agent. The separation of local region and inter region agents contributes to a faster achievement of an optimal model, due to a smaller action space in each agent instead of having a collective bigger one and a more specific reward function, that considers the different impacts that local and outgoing traffic have on the DRL algorithm. The traffic is represented in a traffic matrix as traffic demands, which denote the entire amount of traffic that has to go from the source node to destination node. It is based on this information that the traffic is classified into terminal or outgoing. For the training of the agents, this study uses a mixture of offline and online learning. Firstly, in the offline training stage, the agents are trained with a simulated network that emulates the real one. Secondly, in the online training stage, the learning acquired from the first stage is loaded onto the agents and they are deployed in the real network.

The T-agent is defined as:

- State space: is composed by edge utilizations. Edge statistics take into account traffic volume and route selection to indicate suitable link states.
- Action space: is composed by the paths connecting the origin and destination nodes.
- Reward: the policy is developed based on the state of the local network, always prioritizing actions that keep edge costs low.

The O-agent is defined as:

- State space: similarly to the T-agent, the O-agent's state space is composed by edge utilizations.

- Action space: in the same way as the T-agent, the O-agent's action space is composed by the possible path routes.
- Reward: because this agent attends to outgoing traffic, for a good evaluation of an action it is necessary to take into account its impact in the local and neighboring areas.

For the training process of the agents, the DRL algorithm used was DDPG. This way of training will provide the agents with robustness to the changing traffic patterns, different network configurations from link failures and overall adaptability to a new scenario. This study has shown that using a multi-agent DRL approach with two agents trained with an actor-critic method to manage a broad network scenario perform well in optimizing TE and can improve congestion.

In Zhao, Wu, and Le [26], a MARL method is presented to improve performance in inter-domain routing. As mentioned above, in the Routing section, inter-domain routing as seen in BGP has some limitations such as the inability to take network performance into account when routing, which results in a less than optimal performance. The proposed solution for this issue is that, in the network, each AS has its own agent whose goal is to maximize the average throughput. Each agent uses the following state, action and reward model:

- State space: a tuple with routing information such as starting and ending point of the flow and throughput values of the links connecting to neighbors, for each action on a flow. The state is also conditioned by the degree to which information is shared with the other agents.
- Action space: neighboring agents with an associated probability calculated with a NN
- Reward: average throughput of all the flows for an agent.

Regarding the DRL agent, an actor-critic algorithm is used. In the performed simulations, the throughput in the proposed approach was improved compared to default BGP routing even when increasing the number of nodes, demonstrating the scalability of this strategy.

In Pinyoanuntapong, Lee, and Wang [27], a distributed control approach to TE with a multi-agent model-free framework is proposed to minimize the end-to-end delay TE metric. Reducing end-to-end delay in a complex and broad network is challenging because there are too many variables at play, such as unpredictable network traffic, dynamic network layout and variable links status. All these factors considered together make it difficult to be able to easily control traffic flow while optimizing delay. This approach proposes an alternative to the Q-learning method which has its limitations, such as slow

convergence. The authors define the problem as a multi-agent MDP, and for the solution they propose a multi-agent asynchronous actor-critic architecture, that consists of a set of three learning modules with distinct implemented algorithms for each agent. One module is responsible for policy evaluation, where the action-value modules are calculated as well as the expected reward of each action. In the second module, policy improvement, the expected reward values will be maximized using an actor-critic-executor learning method using the values predicted in the previous module and the action-value data will be updated. The third module, policy execution, is where the knowledge formed in the previous module will be applied to the network, maximizing the reward according to the state it is in. From simulations performed, it is possible to see that a multi-agent distributed TE approach can handle a high volume of data and decrease delay times.

In Lowe et al. [28] the authors explore the application of the MADDPG and compare it to DDPG. The main method is to use centralized training with decentralized execution. This strategy is applied in the actor critic structure by using a decentralized actor and a centralized critic. The centralized critic has access to the other agent's actions and state details Although, during execution, the actor can solely access local information for the learned policies. The authors find that by using MADDPG agents efficient coordination is attained with greater ease.

Table 2.2 summarizes essential aspects of the methods presented above, that were applied to routing scenarios.

Table 2.2: Comparison of multi-agent DRL methods' key aspects

	Multi-region [25]	**Inter-domain routing** [26]	**Multi-agent actor-critic-executor** [27]
Method	Network is separated by areas, each one has two agents: T-agent: incoming traffic O-agent: outgoing traffic	Improve performance in inter-domain routing	Minimize end-to-end delay
Agent type	Decentralized / centralized	Decentralized	Decentralized
Algorithm	DDPG	Actor-critic	Actor-critic-executor
State Space	Edge utilizations: ratio of traffic demand and routing decisions	Start and end nodes; flow information for the agent; traffic load status of neighboring links (number of flows divided by the capacity); neighbor's throughput values for each action	Local network states
Action Space	Previously computed paths	Neighboring agents	Next-hop routers
Reward	T-agent: keeping local edge costs low O-agent: minimizing edge costs of both local and neighboring regions	Maximize throughput for an agent	Minimize delay

3

System Design

The scope of this project is to study the use of DRL and multi-agent DRL in routing scenarios while also analysing its response to a changing network topology.

In this chapter, the modelling of the problem as a MDP (state space, action space and reward function) that is solved by a DRL agent is presented. The model is then used to define the agent's algorithms that will be implemented, to assess what version is the most suitable and has the best response to network changes. The same algorithms will be tested in different topologies, with different characteristics.

3.1 Algorithm and extensions

We consider an actor critic algorithm, MADDPG, that is implemented with minor variations in terms of the centrality of the training, in order to analyze the impact on the model.

DDPG is a resourceful algorithm for our scenario because of its characteristics such as being able to be used in a continuous action space, combining the advantages of actor and critic methods for estimating the policy and value functions for improved learning. Moreover, it has a target network that allows for higher stability when updating the network's weights and experience replay, a feature that allows for a more efficient learning process by eliminating correlation between data that was observed sequentially. The implementation of this algorithm in multiple agents, multi-agent DDPG (MADDPG), is a promising strategy to increase scalability.

Starting with this algorithm as a foundation, modifications were made, regarding the amount of information considered in the critic module. There are two types of critics according to the data that they can use: local critic and central critic. In the local critic scenario, the agent only has access to data observed by the agent itself. In the central critic scenario, the agent has access to data regarding the whole network.

Different variations in the algorithm used in the critic module are also implemented. One implementation is the base version of MADDPG, with a simple Q network and the other is a dueling Q network. Both of these strategies will be explained later in the

document.

3.2 Model Definition

The model structure has learning efficiency in mind, trying to balance relevant network information and data load. The problem is modeled as a Markov Decision Process (MDP) $(S, A, R(s, a))$, with a finite state space S, an action space A, and reward function $R(s, a)$:

- S: The state space is composed by agent observations
- A: The action space aggregates the set of possible actions to take.
- $R(s, a)$: The reward function attributes a reward value to a state $s \in S$ and action $a \in A$, providing feedback on actions and guiding the learning process towards the desirable outcome, which in this case is available link bandwidth.

The following sections explore each of the components in detail.

3.2.1 State space

The state space is defined by an aggregation of elements pertaining to the agent. For this experiment there are two variations of the state: local state and central state.

For the local state, the array is composed by the elements defined in equation 3.1.

$$S_{local} = (\text{NBWs}, \text{D}, \text{C}, \text{BW}_{sent}) \tag{3.1}$$

- NBWs (Neighboring Bandwidths): this field has the available bandwidth in percentage for the nodes directly connected to the agent
- D: Destination for current communication
- C: List of current communications in a node
- BW_{sent}: Bandwidth for the data flow of the packet to be sent

The space allocated for the first element is fixed according to the highest degree of connections a node has in each network topology.

For the central state, the array is composed by the elements defined in equation 3.2.

$$S_{central} = (\text{BWs}, \text{D}) \tag{3.2}$$

- BWs: Available bandwidth for each link in the network
- D: Destination for each agent

The state space is used as input for the agent's actor and critic modules. For the actor module, the policy network, S_{local} is always used, whereas as for the critic module, the value network S_{local} is used in the case of a local critic, while $S_{central}$ is used in the central critic case.

3.2.2 Action space

The action space is the list of options the agent can choose from, when selecting the next move, and it contains the k shortest paths between two routers for a communication. The paths are previously calculated by a Dijkstra algorithm when the environment is being set up. This approach is based solely on the number of hops and does not consider link bandwidth. The agents will adapt to bandwidth availability and optimize it during the training phase with the reward values. The action space, for the k shortest paths, is represented in equation 3.3.

$$A_k = \{a_1, a_2, ..., a_k\} \tag{3.3}$$

This approach, as used in Geng et al. [25], allows for a faster training process of the agents considering the agent only has to choose from a set of values, saving time and resources. One aspect that sets this method apart for training efficiency is that during the training process paths that are unsuitable and would be deemed resource consuming are automatically excluded, either because they could be too long, or simply not bandwidth efficient. Moreover, by iterating through the available nodes, the likelihood of encountering a void path when training is reduced. This method of pre-selection assures that only paths that are regarded appropriate are taken into consideration.

For this experiment, taking into account the number of links in the topologies, and after performing tests with various numbers, no improvements were found from using a higher value, therefore three paths are considered for the action space.

3.2.3 Reward function

The reward is attributed according to the result of an action performed. It is a way to measure success in the experiment and to measurably reinforce actions that drive the learning process in a positive way.

The reward value is calculated based on the impact an action has in the environment, in this case, actions are rewarded according to the remaining available bandwidth in the paths they chose, which corresponds to an optimization goal of maximizing the available bandwidth. For this calculation the node neighbor's available bandwidth is used. Equation 3.4 shows the piecewise function that represents the reward according to the available bandwidth intervals.

$$R(bw) = \begin{cases} 50 & \text{if } bw > 75 \\ 30 & \text{if } bw > 50 \\ 0 & \text{if } bw > 25 \\ -20 & \text{if } bw > 0 \\ -70 & \text{if } bw < 0 \end{cases} \tag{3.4}$$

A separate reward is used for each agent, as opposed to a joint reward, where all of the network's links would be considered for the reward calculations. Some of the reasons that support this approach are that a separate reward offers transparency in the sense that is it easier to understand how a specific action affects the network state because behaviours are looked at independently and the agents can specifically learn from them. It would not be productive for an agent to adjust its policy because of an action taken that is so distant from it that had no impact on its performance

3.3 Multi Agent Deep Deterministic Policy Gradient

3.3.1 Policy Network

The policy network π , in equation 3.5, estimates the best action, that will maximize the reward obtained.

$$a = \pi(S_{local}; \theta) \tag{3.5}$$

The actor takes local state observations S_{local} as input and outputs the most suitable action a for that state, based on the parameters θ of the policy network. The parameters are updated based on the environment feedback for each action taken.

To calculate the network parameters θ, a positive gradient is used, equation 3.6, for the policy network's weight updates to maximize the reward value, for each agent.

$$\nabla_\theta J(\theta) = E\left[\nabla_\theta(\log \pi(S_{local}; \theta))Q^\pi(S, a)\right] \tag{3.6}$$

The optimization algorithm used for the gradient was the Adam optimizer, a widely used method. The update rate is based on the learning rate α.

$$\theta_{t+1} = \theta_t + \alpha \text{AdamOptimizer}(\nabla_\theta J(\theta)) \tag{3.7}$$

3.3.2 Value Network

The value network, represented by the action-value function $Q^\pi(S, a)$, computes the Q values for each action a to evaluate their impact on the environment, represented by the state S. For the value network's weight updates, a gradient descend is used to minimize the loss function.

The optimal policy is calculated with the loss function, in equation 3.8.

$$\mathcal{L}_\theta = E\left[(Q^\pi(S,a) - (r + \gamma Q^\pi(S',a)))^2\right] \tag{3.8}$$

- $Q^\pi(S,a)$ is the value network
- r is the reward
- γ is the discount factor
- $Q^\pi(S',a)$ is the future state estimated Q value

Similarly to the policy network, the Adam optimizer algorithm was used.

3.3.3 Target Network Technique

In the multi-agent architecture, each agent has its own actor-critic network, one online network pair and one target network pair. These pairs of agents are used in the target network technique. Figure 3.1 illustrates the technique's structure.

This method consists of only updating the target network's weights periodically and at a slower rate than the online network's, which are updated during the training process, when interacting with the environment. This method is used to stabilize the learning process, utilizing the target network's weights as a benchmark for updating the online network's weights.

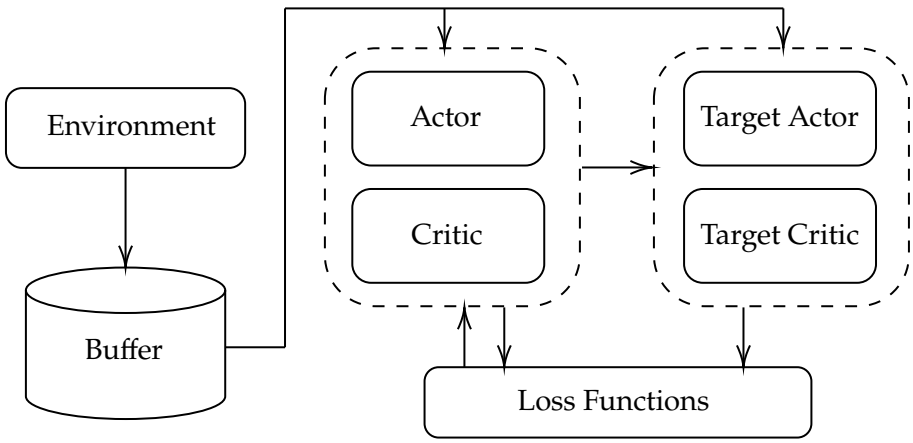

Figure 3.1: Target network technique diagram

3.4 Agent Structure

3.4.1 Policy Network

For the policy function estimation, the actor receives as input the environment state observations and estimates the best action to take according to a policy function.

The state undergoes a linear transformation and it applies a Rectified Linear Unit (ReLu) activation function. That data, then passes though a hidden layer, with another linear transformation and a softmax activation function. It then outputs an array according to the number of actions (N_ACTIONS) to choose from.

Figure 3.2 illustrates the policy network structure.

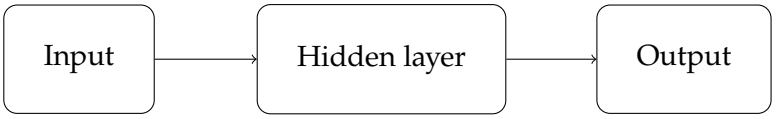

Figure 3.2: Policy network diagram

The input size matches the agent's state, that is composed by the available bandwidth percentage of the node's neighbor links, the destination node, current communications in the link, and packet flow bandwidth value to be sent $S_{local} = (\text{NBWs}, \text{D}, \text{C}, \text{BW}_{sent})$.

The maximum neighboring links value, NR_MAX_LINKS, is set according to the highest neighbor degree of a node and changes accordingly to the topology. That value is acquired by iterating through the nodes in a topology and finding the maximum number of connections a single node has.

The maximum number of active connections, NR_ACTIVE_CONNECTIONS, refers to the maximum amount of packet flows that a link can handle at the same time.

This results in a state space array with the dimensions of [NR_MAX_LINKS + 1 + 2 * NR_ACTIVE_CONNECTIONS + 1]:

- NR_MAX_LINKS: the maximum number of neighbors a node can have
- 1: the destination node for the communication
- 2 * NR_ACTIVE_CONNECTIONS: the number of possible flows that can go though one node multiplied by two to account for the origin and destination nodes
- 1: for the flow bandwidth

Information about the network layers is displayed in table 3.1.

Table 3.1: Policy network layers

Layer	**Type**	**Input**	**Output**	**Activation function**
Input	Linear	S_{local}	-	ReLu
Hidden	Linear	-	-	Softmax
Output	Linear	-	N_ACTIONS	-

3.4.2 Value Network

In contrast to the policy network's input size, that is the same for both the central and local critic cases, the value network's input size changes based on the type of critic. The critic can be local or central, which will reflect on what kind of information the agent has access to.

In the local critic case, the input size is the same as the policy network, because the input is identical, $S_{local} = (\text{NBWs}, \text{D}, \text{C}, \text{BW}_{sent})$.

Whereas for the central critic, where the agent has access to the entire network's link bandwidth and the list of destination nodes: $S_{central} = (\text{BWs}, \text{D})$. The input array has the dimensions of [NR_LINKS + NR_AGENTS]:

- NR_LINKS: total number of links in the topology
- NR_AGENTS: number of agents in the topology

3.4.2.1 Simple Q Network

The simple Q network implementation uses a linear transformation and ReLu activation between the input and hidden layer.

The hidden layer is directly connected to the output layer. By doing so information flow is optimized, as the focus in this implementation is to capture key aspects with low processing complexity.

It outputs one value that estimates the reward for an action in a state.

Figure 3.3 illustrates the network structure.

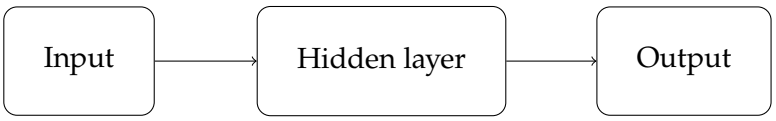

Figure 3.3: Simple Q Network diagram

Information about the network layers is displayed in table 3.2.

Table 3.2: Simple Q network network layers

Layer	Type	Input	Output	Activation function
Input	Linear	$S_{central}$	-	ReLu
Hidden	Linear	-	-	-
Output	-	-	1	-

3.4.2.2 Dueling Q Network

In the dueling Q network scenario, the input layer separates into two for the Value and Q value calculations. This approach was proposed in Wang et al. [29], and exhibited increased performance when compared to other DRL methods.

The layer that calculates the Value uses a ReLu function as activation and outputs one value per state. Whereas the Q value stream uses a softmax activation and outputs a value for each possible action. The Value measures the quality of a state and the Q value indicates how good an action is for that state. The advantage function compares the Q values to the average and by doing so, indicates how much an action is better or worse than the average.

Equation 3.9 illustrates the steps.

$$\begin{aligned} AdvantageFunction &= Q_{values} - Average(Q_{values}) \\ Q &= Value + AdvantageFunction \end{aligned} \tag{3.9}$$

Figure 3.4 illustrates the value network structure with a dueling Q network modification.

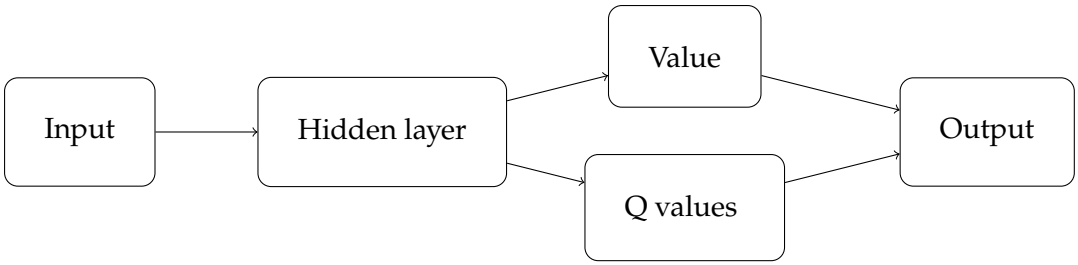

Figure 3.4: Dueling Q Network diagram

Information about the network layers is displayed in table 3.3.

Table 3.3: Dueling Q network network layers

Layer		Type	Input	Output	Activation function
Input		Linear	S_{local} or $S_{central}$	-	ReLu
	Value	Linear	-	1	-
Hidden	Q values	Linear	-	N_ACTIONS	Softmax
Output		Linear	-	1	-

3.5 Training Algorithm

Firstly, the environment is setup to better emulate the behaviour of a real routing network.

For each timestep in each episode, the communication's origin and destination nodes are read from a traffic matrix.

Each agent is going to observe the environment and create its state based on the available bandwidths of the links connection the agent to its node neighbours, the communication destination node, the active communications in the link connecting the agent and the next node, and the bandwidth for the packet flow to be sent.

The agents are trained simultaneously in the same environment. With a shared network environment agents can cooperate from sharing local information with each other. Although, local information sharing is not the case for all the algorithms. With this setup the goal is to achieve an efficient network exploration and effective cooperation between the multiple agents.

An action, previously calculated path, is chosen for each agent based on the current policy and exploration. Each chosen action is saved in a dictionary until all the agents have an action.

The actions are performed, the bandwidth associated with a data packet flow is subtracted from each link bandwidth in the path until it reaches the endpoint.

This will lead to new environment states and the reward is calculated, one separate reward for each agent.

This information is saved in the experience replay buffer, to be randomly selected in the learning process.

The training algorithm is described pseudocode in algorithm 2.

Algorithm 2: MADDPG training algorithm

```
Initialize: agents, replay buffer and environment
    NetworkEngine()
    NetworkEnvironment(NetworkEngine())
    MADDPG(number of agents, parameters)
    MultiAgentReplayBuffer(buffer size, parameters, batch size)
for each epoch do
    for each episode do
        NetworkEnvironment reset:
            Links are back to full capacity
            New communication sequences
        for each timestep in replay buffer batch size do
            Get destinations
            Observe environment
                Get state S_local = (NBWs, D, C, BW_sent)
            Get actions for each state
            for each agent do
                Choose action
                Build actions dictionary
            end
            NetworkEnvironment step (perform actions)
            Observe environment after actions performed
            Store transitions in the replay buffer:
                MultiAgentReplayBuffer store
                  (S_t, A_t, R_t, S_t+1, Critic states, Critic new states)
        end
        if epoch % x then
            Learn(MultiAgentReplayBuffer)
    end
end
```

4

Implementation

This chapter explores the different topology scenarios used to train the agents, the agent implementations and their training in the simulated environment.

The three different topologies used, allow to study the multi-agent models in diverse scenarios as each experience has different characteristics.

4.1 Architecture

The DRL algorithms are implemented in Python, using PyTorch, a framework for ML.

The Agent class has the NN implementations for the actor and the critic modules.

The MADDPG class is where the training loop takes place. This process was described in the previous chapter.

The Multi Agent Replay Buffer class implements the **experience replay** technique. The buffer is central, meaning that it is shared among all the agents and it is circular, where new information overwrites older information. Environment transitions are saved in a memory buffer for later retrieval in a random order for network weight updates. By doing so, sample correlation is eliminated because the agent is not looking at events in a chronological order. This offers learning stability and sample efficiency. It also helps with balancing exploration and exploitation because the agent can learn from both current and previous states.

The Network Environment class encapsulates the environment and implements RL training methods such as, "step", "reward" and "reset". It is implemented using the OpenAI Gym toolkit, that grants access to the environment interaction functions.

The "step" method is executed at each time step. It applies the agent's actions to the environment, transitioning it to the following state. From this interaction state transitions are generated with relevant information for the learning experience.

The "reward" method, attributes a numerical value to an action's outcome, this provides feedback and shapes the learning process to reach a desirable outcome. A reward is attributed to each agent, separately.

The "reset" method initializes the environment back to the base conditions for a new training session. In this project it reinstates links' capacity, resets statistic values and gets the next traffic matrix.

The Network Engine class simulates the routing network behaviour. It interprets the network topology, that is represented in NetworkX, a python library used to manipulate and study networks. It is useful in the context of routing networks, because the edges and links saved in graph form can have attributes and store data. Using the Network Component class the topology information is translated into nodes. The Link class is used to create the edges in the NetworkX topology and assign them properties. It simulates the routing process by forwarding packets through the selected paths, adding and removing communications from a link when they are in course or when they are finished. This class can also introduce changes to the topology links for testing, adding and removing connections. The traffic matrices used for the communications in each topology are generated in this class. Each traffic matrix is generated randomly, assigning 30 destination nodes, for each origin node. Therefore, in each episode, each agent has the workload of sending 30 data packets.

The interactions between the different code modules are illustrated in 4.1

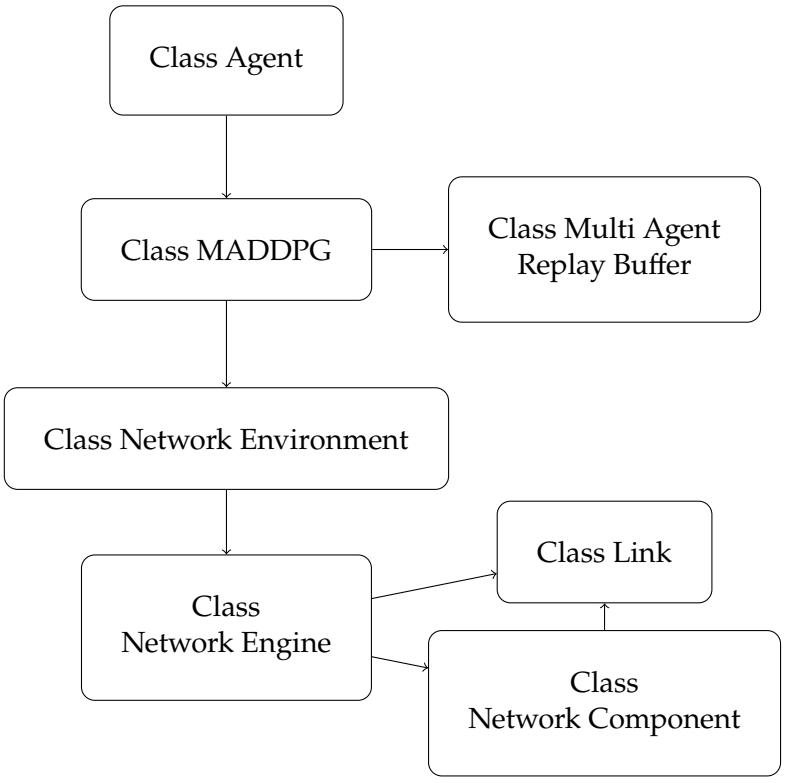

Figure 4.1: Architecture diagram

4.2 Agent parameters

The agent hyperparameters influence how the learning process goes during training.

The learning rate α controls how frequently to update the network's weights.

The discount rate γ influences how future rewards will be considered in current decisions. A higher value means the agent considers less how current actions will impact the future state.

The target network update rate τ affects the frequency of the target network weight updates.

The exploration rate ϵ is related to the ϵ-greedy strategy, where an agent chooses a determined action with ϵ probability. This process is a means for exploration because paths that might otherwise not be taken are selected. This implies that the action chosen may not be optimal, but this process is important to discover new network insights. As the simulation progresses there is a gradual reduction in the ϵ probability with the rate of 0.0001 per epoch. This encourages the agent to exploit actions that are known to have a higher reward rather than unknown ones. This strategy helps with learning efficiency.

The replay buffer stores 1000 transitions and batches of 100 are used for each learning step.

The hyperparameters established for training the agents are specified in table 4.1.

Table 4.1: Parameters

Parameter	**Value**
Learning rate α	0.01
Discount rate γ	0.95
Target network update rate τ	0.001
Exploration rate ϵ	0.3
Replay buffer size	1000
Replay buffer batch size	100

4.3 Topology Scenarios

The various DRL algorithms presented in the previous chapter were tested over three distinct routing network topologies.

The three network topologies used, were: a power law topology with 25 nodes, an ARPANET topology with 33 nodes and a service provider topology with 65 nodes. The algorithms were trained in each of these topologies and then tested to evaluate performance.

4.3.1 Power law topology

Using a NetworkX's library function, `random_internet_as_graph(n)`, a topology with 25 nodes and 41 links was generated. This function creates topologies with the number of nodes specified in the function's argument, in which the distribution of the nodes' degrees follows a power law. The choice of this type of network has to do with the fact that the internet AS topology also follows this degree of distribution. The power law degree distribution means that only a small amount of nodes have a high degree, high number of neighboring nodes, while most of the nodes only have a few connections.

This topology has 15 nodes with a single connection, and 10 nodes with multiple connections. The node with the highest degree has 11 links.Figure 4.2 illustrates the connections between them.

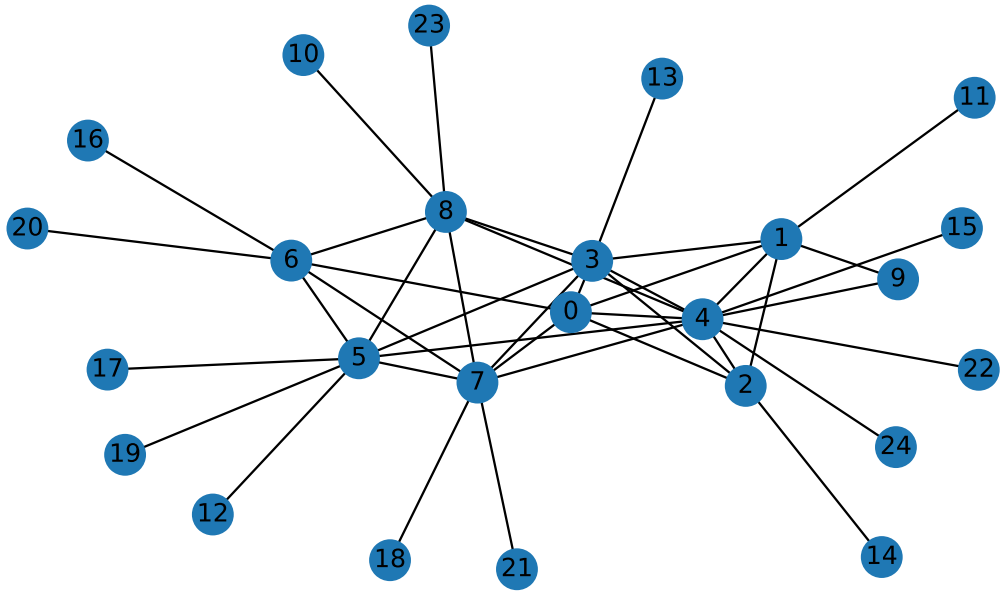

Figure 4.2: Power law network topology

Regarding state space dimensions, the power law topology has:

- S_{local} = (NBWs, D, C, BW_{sent}): [NR_MAX_LINKS + 1 + 2 * NR_ACTIVE_CONNECTIONS + 1] = 33
  - NR_MAX_LINKS = 11. The highest degree of links one node can have.
  - NR_ACTIVE_CONNECTIONS = 10. The maximum number of packets that can be passing through a node
- $S_{central}$ = (BWs, D): [NR_LINKS + NR_AGENTS] = 65
  - NR_LINKS = 41
  - NR_AGENTS = 25

4.3.2 ARPANET topology

The second network topology studied was ARPANET. This topology was the first packet switching network in the early days of the Internet, although it does not follow a power law in terms of node degree. This aspect sets the two topologies apart and from an agent and action space standpoint could be interesting to study.

This topology has 33 nodes, and they are all viewed as switches that can communicate with each other, and has 47 edges, which makes it larger than the previous topology. In this topology, there are 13 nodes with a single connection and 20 nodes with multiple connections. The highest degree node has 6 connections. Figure 4.3 illustrates the ARPANET topology.

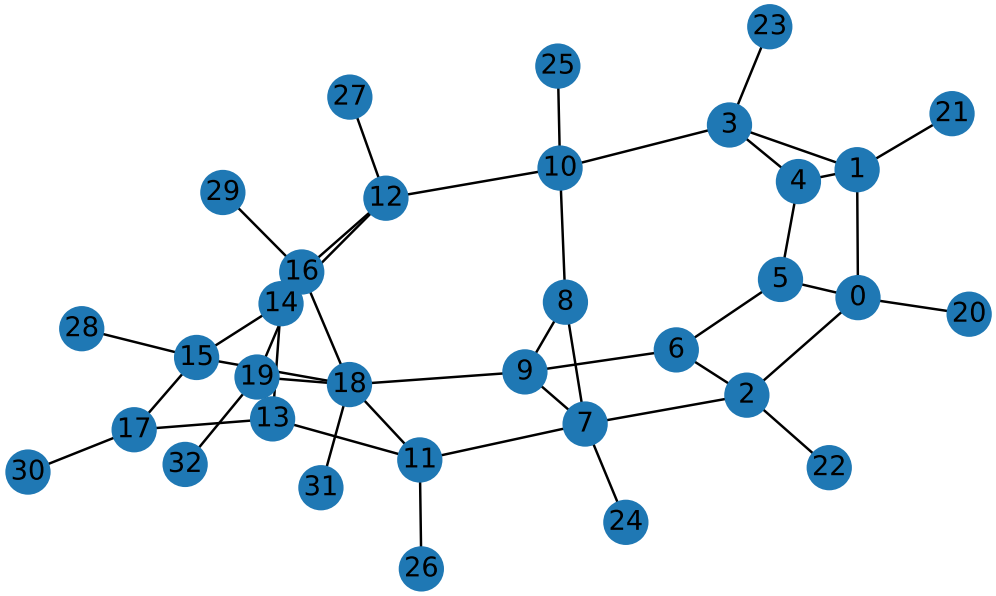

Figure 4.3: ARPANET Network topology

Regarding state space dimensions, the ARPANET topology has:

- S_{local} = (NBWs, D, C, BW_{sent}): [NR_MAX_LINKS + 1 + 2 * NR_ACTIVE_CONNECTIONS + 1] = 28
  - NR_MAX_LINKS = 6. The highest degree of links one node can have.
  - NR_ACTIVE_CONNECTIONS = 10. The maximum number of packets that can be passing through a node
- $S_{central}$ = (BWs, D): [NR_LINKS + NR_AGENTS] = 80
  - NR_LINKS = 47
  - NR_AGENTS = 33

4.3.3 Service provider topology

To test the algorithms in a real case topology, it was used the service provider topology depicted in figure 4.4. This service provider topology has an entirely different structure, where there are various access nodes, that are connected to some intermediate nodes, that connect to an outer node. There are some nodes connected in a ring form, where they are directly connected to just one other node.

Besides from the different network structure, this topology differs from the two previous ones in the sense that only a few nodes can initiate and receive data packets. This type of topology, being from a service provider, separates the nodes that have contact with outside the network from the internal ones, and only those can transmit data packets. The exterior connections are not present in the graph, however they still are considered and specifically, the set of nodes is: $\{7, 11, 12, 20, 21, 29, 30, 37, 38, 46, 47, 55, 56, 64\}$.

This topology has a total of 65 nodes and 85 edges. All of the nodes have multiple connections, with the highest degree being 4. Figure 4.4 illustrates the service provider topology.

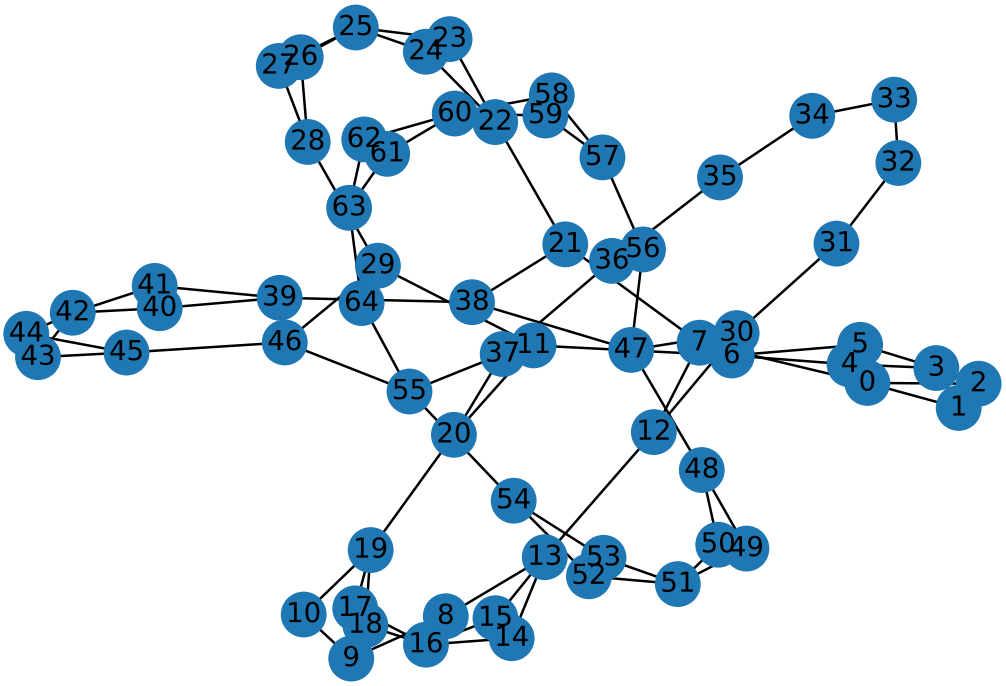

Figure 4.4: Service provider network topology

Regarding state space dimensions, the service provider topology has:

- S_{local} = (NBWs, D, C, BW_{sent}): [NR_MAX_LINKS + 1 + 2 * NR_ACTIVE_CONNECTIONS + 1] = 16
  - NR_MAX_LINKS = 4. The highest degree of links one node can have.
  - NR_ACTIVE_CONNECTIONS = 5. The maximum number of packets that can be passing through a node

- $S_{central}$ = (BWs, D): [NR_LINKS + NR_AGENTS] = 150
  - NR_LINKS = 85
  - NR_AGENTS = 65

4.4 Training Scenarios

In the training scenarios, the following variations of the critic agent were used:

- Simple Q network: the agent estimates value reward from the environment observations. This approach observes underlying associations in the states and actions and can be suitable for various environments, according to their level of complexity, for example a network with intricate relationships, and dimension, which will affect the state space.

- Dueling Q network: the agent can extract more specific correlations for an action and a state because of decoupling the action and state when estimating the best value. This technique isolates the action value for a specific state. This approach suggests better generalization as it delves into more intricate interactions between the states and actions. However, it requires higher computational processing.

Regarding the degree of network information the critic can access, we can have:

- Local critic: the agent can only access information that is directly observed.

- Central critic: this variation aggregates link bandwidth information for all the links in the network topology, providing the critic with a broad overview. This way agents can coordinate between them and account for global impact when evaluating actions.

Table 4.2 provides an overview into the algorithm variations used in the training simulations.

Table 4.2: Training scenarios

Neural Network	**Critic type**
Simple Q network	Central critic
Dueling Q Network	Central critic
	Local critic

4.5 Testing Metrics

For the testing section, two metrics were chosen to evaluate performance: packet loss percentage and average available link bandwidth percentage.

- Packet loss: A packet is considered lost if in its travelling path, it encounters a link that has no available bandwidth. To calculate the packet loss percentage the number of packets lost is divided by the number of packets sent (successfully and unsuccessfully).
- Average available link bandwidth: The average available link bandwidth is calculated along the episodes by averaging the network's links available bandwidths.

These metrics show whether the network's resources are being managed efficiently, as the goal is to have as much available bandwidth as possible to prevent link congestion that would lead to packet loss.

The traffic matrices used during testing were new and not used during the training process.

4.6 Testing Scenarios

To study how the testing metrics behave in different settings, four testing scenarios were developed.

The first test is performed in the original network topology, used during training.

To study the performance of each algorithm variation in a dynamic network, the agents were tested over different versions of the original topology. To modify the original topology, link failures and new connections are simulated. These changes makes it possible to analyze the impact that normal network operation events have in performance and if continuously training the agent in the new environment can contribute to improvements..

4.6.1 Simulating link failures

To test the behaviour of agents in the presence of link failures, links are removed from the topology.

Nodes that have a single connection are excluded from the pool of nodes that can have a link erased. This prevents them from being isolated from the network, as the goal is to simulate failures with alternative paths, rather than downright blocking communications, which would just lead to packet loss.

The challenge in this experience is that the state size needs to be the same as in the scenario the agents were trained in, and by removing links, the state size is changed. Therefore, instead of removing the link from the network topology, which would change state input dimensions for the DRL agents, the link's bandwidth capacity is changed to 0 to reflect the absence of a link, in the state vector.

For this project there are three scenarios: in "Scenario 1" one link is removed, in "Scenario 2" two links are removed, and in "Scenario 3" three links are removed.

4.6.2 Simulating new node connections

To test the agents behaviour in a different topology from the one used in training, links are randomly and progressively added. This experience aims to explore how the agents adapt to new routing paths, and if the learned policies can be transferred onto a different topology with the same number of links.

Since the size of the state must be maintained, the number of links throughout the experiences must stay consistent. To achieve this, as new network edges are added, existing ones are previously removed.

To prevent isolating network nodes, links that connect single nodes are not removed. Similarly, the maximum number of connections a link has in the original topology has to be maintained, which is important because this number is used in the state space definition, edges are only added to nodes who will stay below that number. Both of the links, removed and added, are randomly chosen from the pool of suitable edges. This process serves the dual purpose of balancing state size and also introduces more diversity into the original topology beyond what adding just one link would offer.

There are three link addition scenarios: "Scenario 1" where one link is added, "Scenario 2" where three links are added, and "Scenario 3" where six links are added. In each scenario, for each link added, another link is removed.

In contrast to the previous experiment, the amount of changed links differs. The aim of simulating new node connections is to analyse the behavior of the agents in a slightly different topology than the one used for training, hence the need to change more links. In the previous experiment, eliminating the same amount of links would just lead to poor network performance and no meaningful conclusions would be drawn.

This experience will only be performed in the power law and ARPANET topologies. Because in the service provider topology, considering its disposition, the addition of random new links would not reflect a real world scenario.

4.6.3 Incorporating continuous learning

Another feature added while testing is the ability for the agents to continuously learn during the evaluation process.

Since in some testing cases, the topology being tested on has some differences to the one the agents were trained on, continuously updating the neural network's weights is a way to adapt the agents to a different environment and continuously improve their performance.

The goal of this experiment is to see how quickly the agents can adapt to unseen network conditions and to minimize its negative impact in the regular network behaviour.

These results will be compared to the link removal and addition experiences to assess performance.

5

Results

In this chapter, the experiences obtained results are presented. Starting with the results from the training of the various agents, showing the reward evolution during training, and then proceeding to the testing of the trained agents.

5.1 Training

The traffic matrix used to train the agents, a file with the source and destination nodes for the communications, was the same for all groups of experiments in each topology.

5.1.1 Power law topology

Figure 5.1 displays the rewards over training in the power law topology.

The epochs have been grouped into batches of 10 to minimize visual noise. Before batching, the reward per epoch in the y axis is calculated as the average reward per agent.

The local critic dueling Q network solution does not significantly improve over time, proving that restricting the critic information access to only local data is not effective for learning in a multi agent scenario. This solution proves to be ineffective despite the usage of the dueling Q network.

The simple Q network has a faster convergence time. This is expected, considering the simple Q network features a simpler architecture than the dueling Q network and can learn faster. It could be also due to a more efficient environment exploration in the beginning stages of training, although, this can lead to overfitting.

The central critic dueling Q network solution starts off as the lowest in reward but over time gradually increases and surpasses the simple Q network. The combination of a critic that can access a wider scope of information with the more sophisticated value estimation. Exploration is more successful with the dueling Q network because, by separating the value from the action in the critic module, it can specifically understand which specific actions will lead to a higher reward.

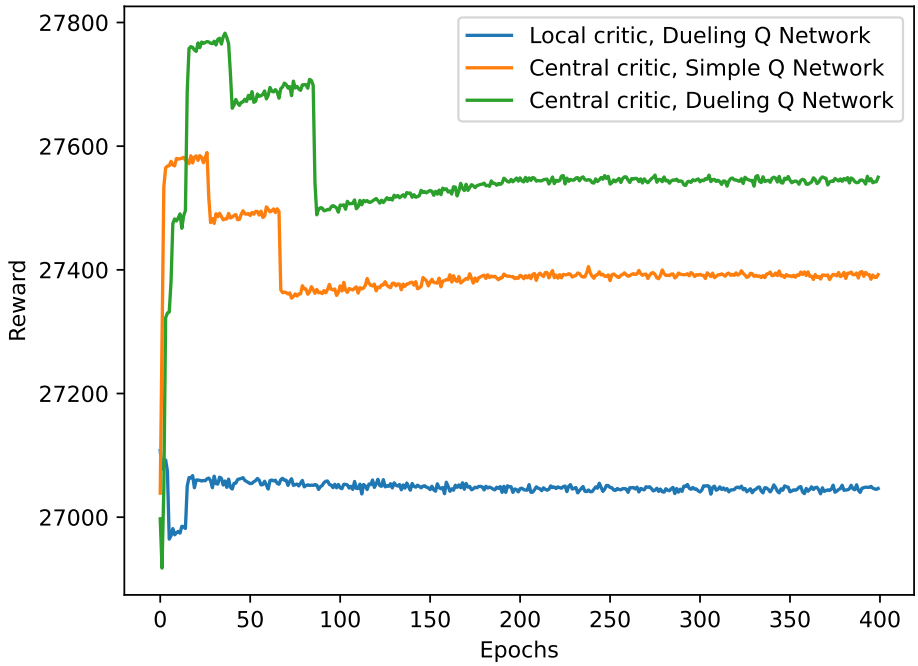

Figure 5.1: Training in power law topology

5.1.2 ARPANET topology

Figure 5.2 shows the results for training the ARPANET's topology agents. This topology has 33 agents, which makes it larger than the previous one, plus more nodes have higher connectivity. The epochs are grouped in batches of 10 to minimize visual noise.

In this experiment, the rewards per epoch are lower due to the increase of nodes with a similar number of links. Despite the slight increase in link number in the ARPANET topology, 45 links, compared to the 41 links in the power law topology, these additional links connect a larger number of nodes. With more nodes sharing the same links, there will be a greater strain on network resources, thus the available bandwidth per link will reduce. This will reflect in the reward values attributed to the agents, as they directly correlate to that metric.

The local critic dueling Q network starts with the highest rewards, but quickly decreases and becomes the lowest performing solution. This suggests that the agent was exploring the environment, choosing actions that had a high reward as an outcome. However, as the agents started exploiting those high reward strategies, no improvements were made. The decreasing rewards could also be due to the critic having only a partial view of the network, so when it estimates action values it does not have the entire topology into consideration and does not allow for cooperation.

The central critic with the simple Q network starts off with the lowest rewards, but gradually increases as it interacts more with the environment, learning from it until

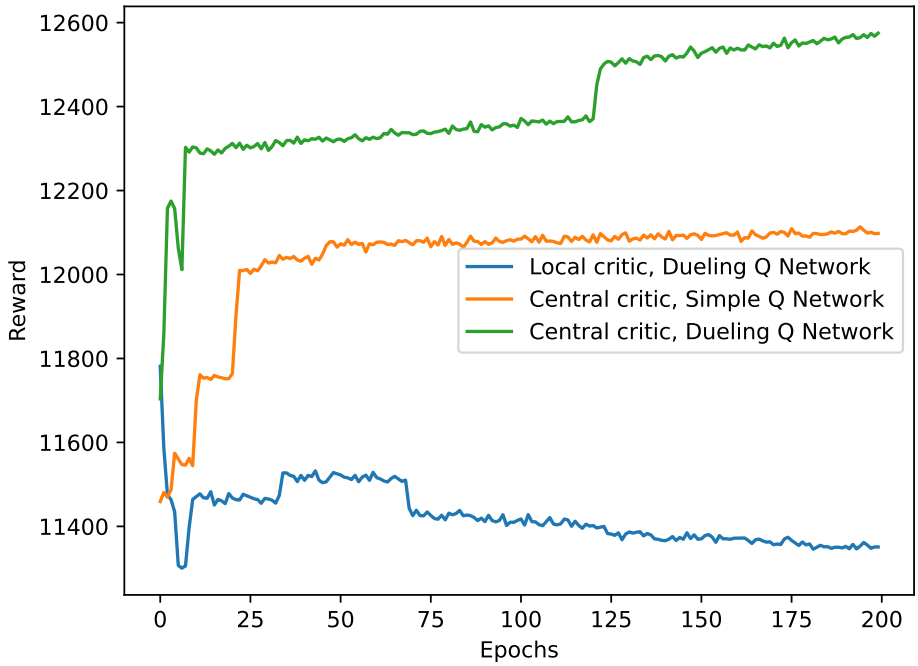

Figure 5.2: Training in ARPANET topology

eventually stabilizing.

The central critic dueling Q network training process begins with rapid growth, as it is exploring the environment successfully. This solution combines the broad view of the network of the central critic with the precise action values calculation of the dueling Q network. After the initial growth, the learning stabilizes until the rewards increase again and continue to show improvements.

5.1.3 Service provider topology

Training in the service provider topology shows that the local critic dueling Q network implementation is the most suitable for a topology of this nature, even with the agents having a narrower perception of the global network state.

This topology, contrarily to the previous topologies where various nodes have high connectivity, has an homogeneous and low node connection degree, where the majority of nodes have from two to three connections, and the rest have four connections. In some topology areas, nodes form closed loop, or ring sub-topologies, which constrains the communication flow to a single path. Sometimes that path may start as a ring, diverge into two nodes, and go back to a ring structure. Due to that, there is not a broad choice of paths for the agents to choose from, that is why the local critic better adapts to this scenario, as general network state will not have impact in a relatively isolated path. As a result, the central critic does not surpass the local critic in performance, contrarily to the

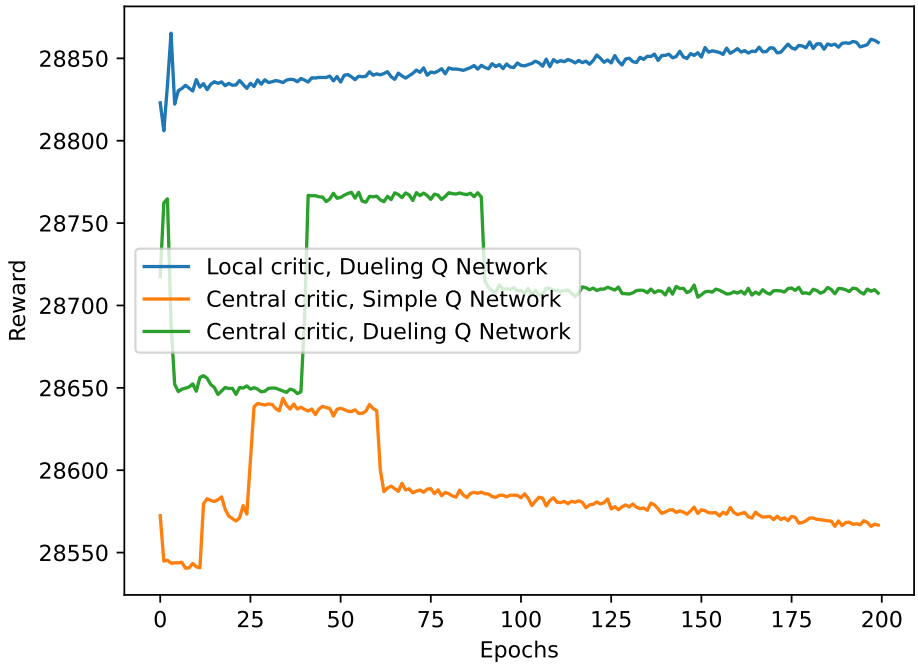

Figure 5.3: Training in service provider topology

power law and ARPANET topologies, where having a broad network overview proved to be a successful technique.

In this topology there is also a dominance in performance by the dueling Q network implementations.

Furthermore, unlike the previous topologies, the agents do not show significant growth over time. The presence of various rings in the node connections can restrict the paths with interdependencies, which constrains the communication flow and makes it challenging to optimize routing.

With all things considered, the local critic solution is the most favorable option as it achieves higher rewards and has a faster processing time.

5.2 Testing

After training, the agents' performance was tested by evaluating two metrics: packet loss and average available bandwidth.

5.2.1 Power law topology

5.2.1.1 Link failures

The algorithms used in the training process are evaluated over four different scenarios: the original network with no topology changes, and Scenarios 1 through 3, where one link

is sequentially removed from the topology.

Each of these experiences is performed twice, the first time the agents retain the NN weights from the training phase, and in the second experience there is a continuous learning process, where the weights are updated. The links are removed randomly in each experience, therefore the links selected to be removed may not be the same across experiences. The experiences were performed multiple times, and the results are an average, so various removed links are considered, as performance may be impacted more by some connections than others. The algorithm considers node connection degrees so there are no isolated nodes.

The results of these experiences are displayed in tables 5.1, 5.2 and 5.3.

Table 5.1: Central critic simple Q network algorithm test when removing links in the power law topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	1.83%	1.83%	73.76%	73.76%
Test Scenario 1	7.75%	7.86 %	72.39 %	72.93%
Test Scenario 2	12.38%	10.29%	71.04 %	71.12%
Test Scenario 3	12.28%	13.59 %	71.05 %	69.66%

The simple Q network solution testing results presented in table 5.1 shows an expected increase in packet loss, as more link are removed. Packet loss improves with continuous learning in Scenario 2. Packet loss does not improve in Scenario 1 and neither in Scenario 3. This could be due to the random links removed being the unfavorable choice of removing high connectivity links or due to the agents trying to optimize available bandwidth.

Overall, available bandwidth percentage across the network links slightly increases with continuous learning, except in Scenario 3, where it worsens. In this scenario, packet loss worsens too, this indicates that the links removed in this experience were crucial to network performance, because the trend observed in the previous scenarios was of improved performance with continuous learning.

For the local critic dueling Q network solution, the testing results are displayed in table 5.2. As expected, with the first link removal, the results shows a degradation in network performance, as the packet loss percentage increases, along with the decrease of available bandwidth. Although, as the learning process progresses, the algorithm adapts to changes in the network and improves results. Average available bandwidth also increases with continuous learning for all the scenarios.

Table 5.2: Local critic dueling Q network algorithm test when removing links in the power law topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	2.72%	2.72%	73.39%	73.39%
Test Scenario 1	6.25%	5.18%	71.52%	71.75%
Test Scenario 2	9.20%	8.76%	69.92 %	70.33%
Test Scenario 3	12.59%	8.76 %	68.39 %	70.34%

Overall the local critic dueling Q network solution responds positively to continuous learning even with topology changes.

Table 5.3: Central critic dueling Q network algorithm test when removing links in the power law topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	1.57%	1.57%	73.78%	73.78%
Test Scenario 1	6.60%	6.13%	72.39%	72.48%
Test Scenario 2	8.76%	8.09%	70.57%	70.60%
Test Scenario 3	12.38%	11.76%	69.27%	69.26%

The central critic dueling Q network solution testing results are presented in table 5.3. This algorithm had the best performance during training, achieving the highest rewards.

The results show improvements for packet loss values with the continuous learning technique. Likewise, the average available bandwidth percentages also increase while learning in the two of the link removal scenarios, although not significantly. In Scenario 3, despite the packet loss improvement, average link available bandwidth stays the same.

Figure 5.4 displays a comprehensive view of the data for the packet loss percentage, presented in tables 5.2, 5.3 and 5.1. It also displays the testing results from a shortest path simulation, where there are no agents involved and the routing is done based on the shortest paths defined in the environment setup.

The graph compares the packet loss percentage values across the different experiences. To highlight differences in the data, the y axis of the graph was adjusted to a fifth of the original scale, as the results can be better visualized with a narrower viewing window.

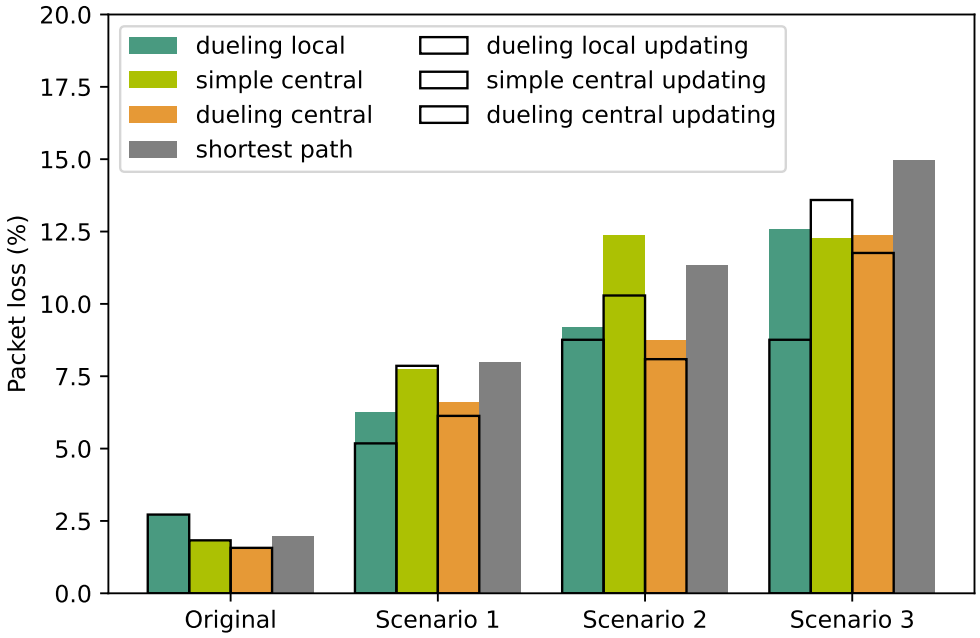

Figure 5.4: Packet loss when removing links in the power law topology

Except for the test in the original network, without link changes, the dueling Q network implementations have overall lower packet loss values compared to the simple Q network one. This shows that this implementation in the critic module adapts better network changes, such as link failures.

In the original network test, the central critic implementations have lower packet loss values compared to the local critic one. Based on these results, the local critic, where the agents only have a partial view of the network during the training process, is not the best option to prevent packet loss in the unchanged topology scenario.

In the original topology, the central critic dueling Q network solution has the best packet loss performance. This is expected because in the training phase results this implementation also had the highest rewards. The central critic simple Q network and local critic dueling Q network came second and third, the same as the training results.

This scenario has consistent results regardless of the learning process, continuous learning or not, which is expected, because the topology is the same where the original training process was done.

Throughout the edge removing scenarios, Scenarios 1, 2 and 3, packet loss increases proportionately to the number of links that are removed per experience, considering that there is the same amount of data packets to transmit and less available paths. The graph clearly shows that even one link loss can disrupt the normal network behaviour, and double the amount of packets lost.

This behaviour is expected in a topology of this nature. Even though the edges of nodes that have multiple neighbors make up 63% of the connections, considering that in the event that even one link becomes unavailable, either because of network problems or

because it was removed, the remaining links have to accommodate the packets that were supposed to go through that path. This is aggravated if the link, that is randomly chosen, happens to be one of the most central, where the traffic demand is higher. The remaining paths will reach their capacity quicker, especially in the third scenario, where there is a significantly higher packet loss when compared to the original network.

Comparing all of the solutions with the shortest path routing shows that they all perform better than the shortest path and adapt better to network changes. This difference is even more noticeable in the continuous learning test, where, overall, all of the DRL algorithms show packet loss improvements. This shows that using multiple agents with DRL techniques provides the agents with resilience to network changes when links become unavailable. Furthermore, using a central critic with a dueling Q network is the best option for this type of network topology.

5.2.1.2 New node connections

To assess how the trained agents perform when operating in changing network conditions, after each testing epoch, a number of links is changed in the network. In the first experience no links are added, in "Scenario 1" one link connecting previously separated nodes is added, while simultaneously removing an existing link to maintain the state size. In Scenarios 2 and 3, two and three more links are incorporated into the changing network. As a result, in the second scenario, the experience has three new links compared to the original topology, and in the third scenario it has six.

Tables 5.4, 5.5 and 5.6 display the testing results for changing network links.

Table 5.4: Central critic simple Q network algorithm test when changing links in the power law topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	1.83%	1.83%	73.76%	73.76%
Test Scenario 1	2.09%	2.72%	73.98%	73.66%
Test Scenario 2	2.60%	2.72%	74.15%	73.46%
Test Scenario 3	3.47%	2.47%	73.30%	73.42%

The performance results while adding new node connections with the central critic simple Q network algorithm in table 5.4 show an overall slight increase in packet loss when continuously learning, except for Scenario3, where it decreases by 1%. Average available bandwidth stayed about the same throughout the experiments. Overall this solution adapted well to the new paths, as packet loss performance did not reduce excessively.

Table 5.5: Local critic dueling Q network algorithm test when changing links in the power law topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	2.72%	2.72%	73.39%	73.39%
Test Scenario 1	2.09%	2.72%	73.14%	73.10%
Test Scenario 2	5.30%	2.34%	72.42%	73.16%
Test Scenario 3	3.85%	6.13%	71.91%	72.10%

Table 5.5 displays the test results for the local critic dueling Q network algorithm when changing node connections to the original topology. In this experience, both packet loss and average available bandwidth improves when continuously learning in Scenario 2. In Scenarios 1, both packet loss and average available bandwidth worsen when continuously learning. In the last scenario, packet loss also increases, and available bandwidth improves slightly.

Table 5.6: Central critic dueling Q network algorithm test when changing links in the power law topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	1.57%	1.57%	73.78%	73.78%
Test Scenario 1	3.23%	2.34%	73.42%	73.51%
Test Scenario 2	4.46%	4.58%	72.50%	72.76%
Test Scenario 3	5.06%	9.42%	72.39%	72.08%

Table 5.6 shows the results for the central critic dueling Q network algorithm testing. In Scenario 1, packet loss improves when continuously learning, and bandwidth also slightly improves. In Scenario 2, packet loss worsens, but not significantly, even though available bandwidth slightly improves. In Scenario 3, despite the learning process, packet loss and available bandwidth worsen. This may be linked to agents trying to discover the best policy to optimize average available bandwidth.

Figure 5.5 displays the information represented in the tables above as well as the shortest path implementation.

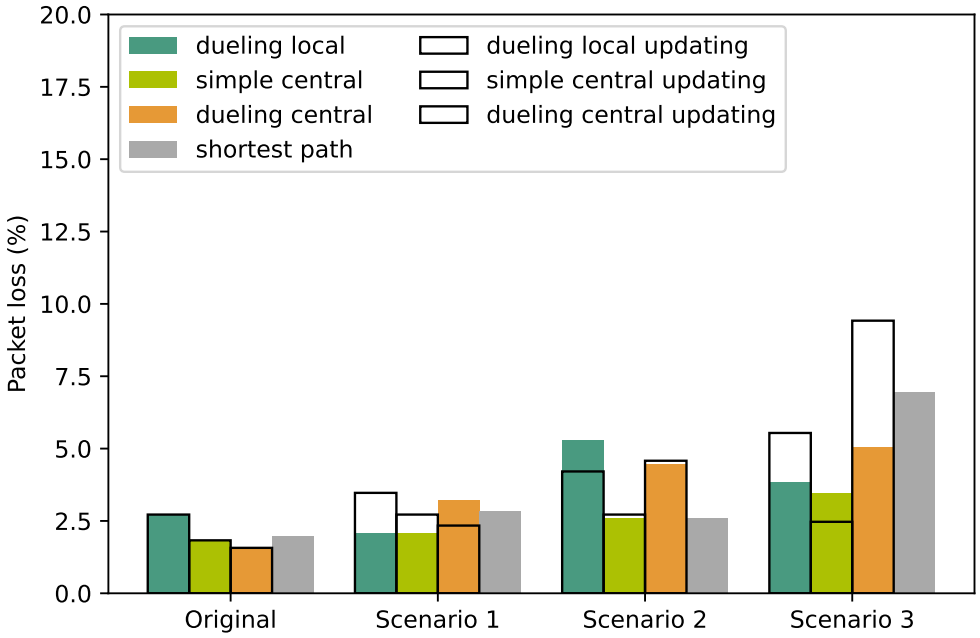

Figure 5.5: Packet loss when changing links in the power law topology

In the original network test, the central critic with the dueling Q network remains the best performing solution compared to the other algorithms. Although, when switching links, the simple Q network algorithm is the fastest to adapt and provides the best results. The faster convergence time is is due to the less complicated critic network architecture that produces faster results.

The local critic solution has higher packet loss than the shortest path algorithm when testing in the original network. Although, over time, with the exception of Scenario 2, the DRL algorithms perform better than the shortest path. This is more noticeable in Scenario 3, where six links differ from the original network, and the DRL algorithms have better performance, especially the central critic with the simple Q network. When changing topology links, the agents still have the learned policies for the original topology, so even with the same number of links available it is expected for performance to not be the same.

5.2.2 ARPANET topology

This topology training results showed substantially lower rewards than the previous topology, so it is expected higher packet loss values and lower available bandwidth.

This occurred due to an increase in the number of nodes in the topology, compared to the power law topology, which lead to more packet flows. As the number of links did not increase proportionately to the number of nodes, this topology experiences a higher traffic load, which resulted in link congestion. Given that the rewards in the training phase are a reflection of the available link bandwidth, the lower values correlate with the link congestion.

5.2.2.1 Link failures

Link failure test results for the ARPANET topology are displayed in tables 5.7, 5.8 and 5.9.

Table 5.7: Central critic simple Q network algorithm test when removing links in ARPANET topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	5.62%	5.62%	68.71%	68.71%
Test Scenario 1	17.77%	14.51%	68.81%	67.64%
Test Scenario 2	26.72%	16.95%	69.74%	66.49%
Test Scenario 3	26.72%	25.79%	69.74%	67.48%

Table 5.7 shows the results for the central critic simple Q network solution. In this solution, even one link removed significantly downgrades packet loss performance, as the traffic flow stays the same for the new smaller number of paths, a lot of congestion is created in high traffic paths. Continuously learning shows a improvement in packet loss, but not in average available bandwidth.

Table 5.8: Local critic dueling Q network algorithm test when removing links in ARPANET topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	6.95%	6.95%	67.97%	67.97%
Test Scenario 1	11.05%	13.76%	66.69%	67.58%
Test Scenario 2	16.53%	20.80%	66.49%	66.57%
Test Scenario 3	20.48%	24.71%	65.62%	65.38%

Table 5.8 shows the test results for the local critic dueling Q network solution. In this implementation, continuously learning does not improve packet loss results. This could be due to the exploration in the agents, making them take actions that are not optimal to study the environment, leading to an increase in packet loss. Although, average available bandwidth values very slightly increase in Scenarios 1 and 2, which indicates that the agents were trying to find the best policy to improve available bandwidth.

Table 5.9: Central critic dueling Q network algorithm test when removing links in the ARPANET topology

	Packet loss		Average available bandwidth	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	6.34%	6.34%	69.29%	69.29%
Test Scenario 1	10.57%	7.30%	68.45%	68.31%
Test Scenario 2	14.29%	15.46%	67.51%	67.29%
Test Scenario 3	19.05%	20.61%	67.45%	66.21%

Table 5.9 shows the test results for the central critic dueling Q network solution. Although packet loss improves when training in the test Scenario 1, in the following scenarios the values increase slightly. Similarly to the previous experience, the agents could be choosing exploratory actions despite average available bandwidth not improving.

Figure 5.6 shows the comparison between the all test values and the results of a shortest path algorithm.

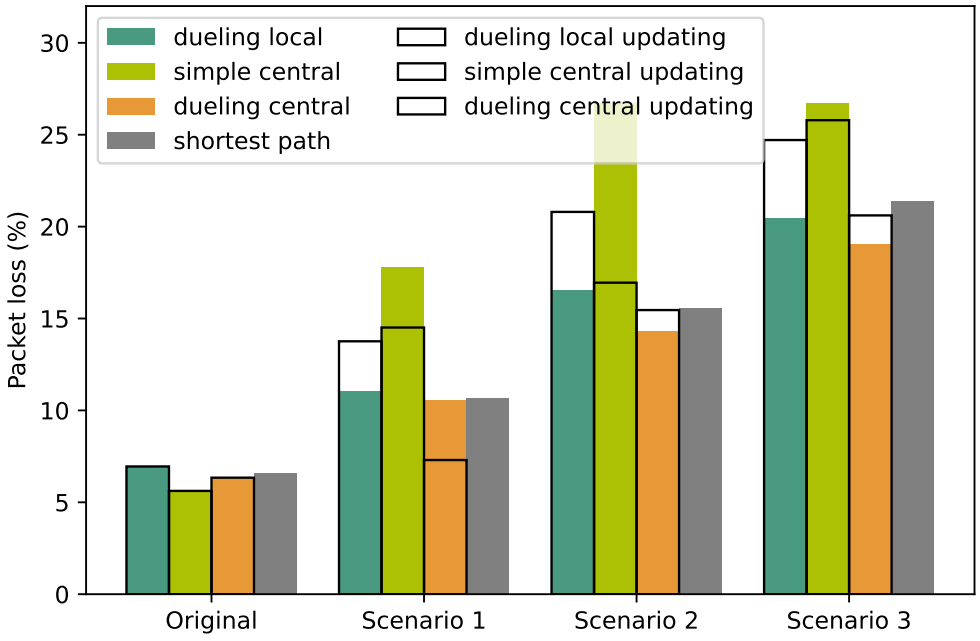

Figure 5.6: Packet loss when removing links in ARPANET topology

In the unchanged network topology, the simple Q network solution has the lowest packet loss despite not achieving the highest rewards during training. The central critic dueling Q network solution, which had the highest rewards during training has the second

best performance during testing in the original topology. The local critic solution had higher packet loss than the shortest path algorithm.

Although, when links are consequently removed, the central critic dueling Q network has the best response. With continuous learning in the simple Q network, the agents quickly respond to changes, but in the dueling Q network solutions they do not adapt as quick suggesting that they explored the new environment instead, with the exception of the central critic in Scenario 1.

5.2.2.2 New node connections

Tables 5.10, 5.11 and 5.12 display the results of the tests when changing network links.

Table 5.10: Central critic simple Q network algorithm test when changing links in the ARPANET topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	5.62%	5.62%	68.71%	68.71%
Test Scenario 1	7.22%	8.16%	69.67%	68.58%
Test Scenario 2	7.65%	4.35%	69.08%	69.06%
Test Scenario 3	13.46%	5.08%	68.23%	66.84%

For the central critic simple Q network solution employed in the ARPANET topology, the testing results in table 5.10 show that when changing a few links there is a slight performance decrease. When more links are changed, the performance decline becomes more noticeable. Although, with continuous learning, the packet loss values attenuate in Scenarios 2 and 3. Regarding the average available bandwidth, the values do not improve.

Table 5.11: Local critic dueling Q network algorithm test when changing links in ARPANET topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	6.95%	6.95%	67.97%	67.97%
Test Scenario 1	9.84%	6.60%	67.44%	67.53%
Test Scenario 2	8.42%	8.33%	66.85%	67.08%
Test Scenario 3	6.25%	13.84%	66.41%	65.44%

Table 5.11 shows the test results when links are added in the ARPANET topology with the local critic dueling Q network algorithm. Overall continuing to update the network weights while training was successful despite the Scenario 3 experiment, where packet loss increased. For test Scenarios 1 and 2, packet loss values decreased, and available bandwidth increased, even if not significantly.

Table 5.12: Central critic dueling Q network algorithm test when changing links in the ARPANET topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	6.34%	6.34%	69.29%	69.29%
Test Scenario 1	6.25%	5.62%	68.71%	69.26%
Test Scenario 2	6.34%	4.35%	68.43%	69.32%
Test Scenario 3	7.74%	5.08%	68.05%	68.27%

Table 5.12 shows the test results for the central critic dueling Q network in the ARPANET topology. Comparing the test in the unmodified topology and the Scenarios 1 and 2, there is no significant change in packet loss values, instilling this solution as an effective choice for topology changes. Moreover, when continuously learning the packet loss values improve, as well as available bandwidth percentages.

Figure 5.7 demonstrates the testing results when changing network links for all the solutions with the addition of a shortest path algorithm.

In general, continuously learning improves the agent's adaptability to network changes, although these algorithms adapt well even without it, as it is evident from the results. With more links are changed from Scenarios 1 through 3, packet loss values do not increase proportionately.

Overall, the DRL algorithms perform better than the shortest path algorithm, except for the tests in the original topology, without link changes, and the first scenario, where the local critic had higher packet loss.

The solution with the lowest packet loss percentage is the central critic with the dueling Q network, this solution also shows faster adaptability to learning and optimizing the new network disposition.

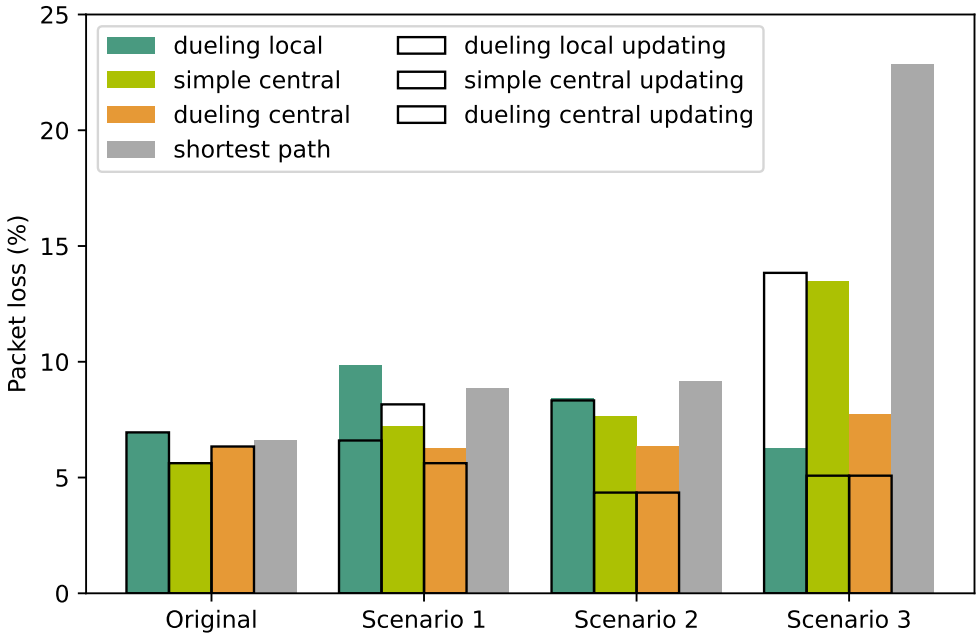

Figure 5.7: Packet loss when changing links in ARPANET topology

5.2.3 Service provider topology

5.2.3.1 Link failures

Tables 5.13, 5.14 and 5.15 show the test results for the link removal experiences.

Table 5.13: Central critic simple Q network algorithm test when removing links in service provider topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	0.02%	0.02%	56.90%	56.90%
Test Scenario 1	10.83%	4.11%	55.68%	56.20%
Test Scenario 2	20.00%	10.54%	56.33%	55.59%
Test Scenario 3	22.87 %	12.77%	55.71%	54.90%

Table 5.13 shows the results for the central critic simple Q network solution. The packet loss is very low in the original topology, although average available bandwidth is not as high as in the previous topology experiences. This topology has low packet loss values because its original node disposition with few connections naturally distributes the traffic load along the network paths.

As expected, performance decreases along with the increase of number of links removed, although its decrease is significant throughout the various tests. Continuously learning significantly improved the packet loss values, even though that was not the case for the average available bandwidth.

Table 5.14: Local critic dueling Q network algorithm test when removing links in service provider topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	0.01%	0.01%	57.36%	57.36%
Test Scenario 1	2.10%	4.76%	56.70%	56.69%
Test Scenario 2	3.78%	7.59%	55.03%	56.00%
Test Scenario 3	7.59%	12.50%	55.37%	55.47%

Table 5.14 shows the results for the local critic dueling Q network test. This solution had the highest reward values during the training phase, which translates to the tests, as it is reflected by the low packet loss values. In the third link removal scenario, packet loss is higher as the impact of three links being removed affects performance.

The continuous learning technique was unsuccessful in this experiment, it seems that the agents took an exploratory approach, to improve bandwidth, and packet loss increased.

Table 5.15: Central critic dueling Q network algorithm test when removing links in service provider topology

	Packet loss		**Average available bandwidth**	
Scenario	**Not updating**	**Updating**	**Not updating**	**Updating**
Test Original	0.01%	0.01%	55.17%	55.17%
Test Scenario 1	3.45%	1.06%	56.48%	56.47%
Test Scenario 2	10.54%	3.11%	55.98%	55.81%
Test Scenario 3	14.64%	3.11%	55.35%	55.81%

Table 5.15 shows the test results for the central critic dueling Q network agent implementation. Continuously learning worked well with this implementation, leading to packet loss improvements. However, available bandwidth stayed about the same.

Figure 5.8 shows the various experiments results along with a shortest path algorithm.

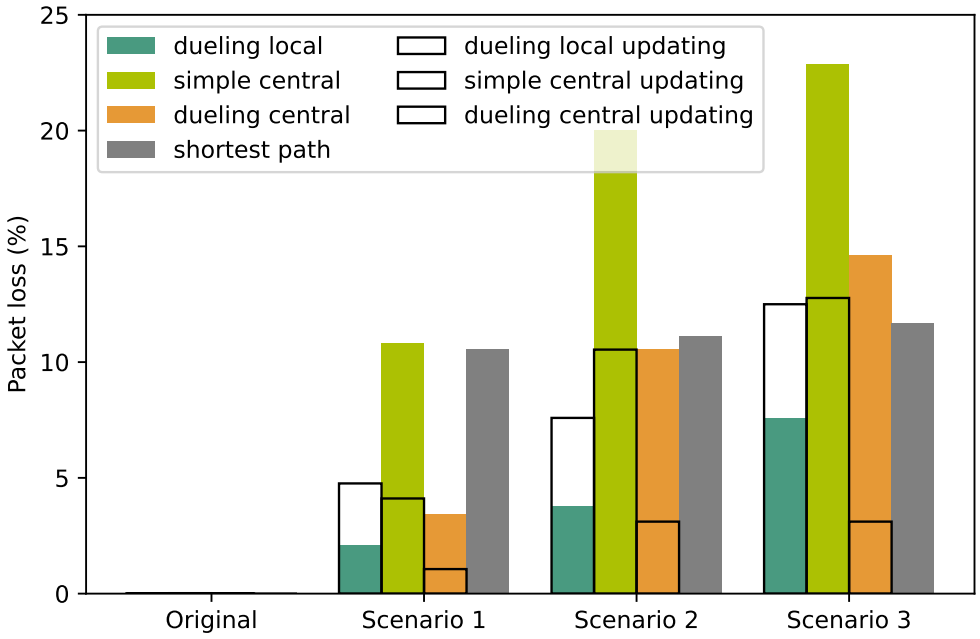

Figure 5.8: Packet loss when removing links in service provider topology

Packet loss is very low for all of the algorithms when testing in the original topology and does not show up in the graph scale. The network structure can handle traffic load efficiently leading to very low packet loss. Although, as seen in tables 5.13, 5.14 and 5.15, the average available bandwidth for the original topology is around 57%, as opposed to the higher values in the power law and ARPANET topologies. The service provider link disposition contributes to this because of the ring sub-topologies. Since nodes are connected in this way, the routing paths are longer, causing the data packet flows to go through more links, and despite the low packet loss, overall link bandwidth reduces. This does not happen in the other topologies because of higher interconnections of the nodes, which will lead to more diverse and direct paths, allowing better traffic flow distribution and more available link bandwidth.

The DRL algorithms perform well against the shortest path. The best performing algorithm is the one that had the highest rewards during training, the dueling Q network with a local critic. The other algorithms also performed according to their standing in the training phase. Although, contrarily to the central critic solutions, the local critic algorithm did not improve performance while continuous learning in the subsequent epochs. The central critic solutions had a much quicker adaptation than the local critic, while learning in the presence of unavailable links for the same number of epochs.

6

Conclusion

6.1 Final remarks

This dissertation explores how MADDPG DRL algorithms perform when faced with topology changes. The topic of scalable DRL based routing is relevant nowadays, considering systems handle great amounts of data and they tend to evolve to more complex configurations. This demands for a dynamic network structure that can support topology changes within the same environment. When the topology changes due to an error, or a new configuration, to minimize negative impact it is necessary to have a system capable of adapting and overcoming that obstacle.

Furthermore, using the multi-agent approach appears to be the most effective method for managing routing and adapting to changes in complex networks because it separates the computational load across multiple agents.

A MADDPG DRL architecture was proposed to deal with these challenges. Some variations were used in the critic module regarding information access, either centralized or distributed, and the in NN architecture, either using a simple Q network or dueling Q network, a more sophisticated approach.

The solutions were implemented and tested over three different network topologies. To evaluate their performance, tests were done with link failures and the presence of new links that were not seen during training. While testing packet loss, performance declined in the presence of network changes, and the agents took some time to re-learn and recover. After assessing the algorithms' test results over the various scenarios, the centralized critic with the dueling Q network showed overall the best adaptability for link failures and changes. Although, in the case of topology changes, removing existing links and adding new ones, for the power law topology, the central critic with the simple Q network solution showed the best results. In the service provider topology, a topology that has very specific node relationships and disposition, the local critic with the dueling Q network showed the best performance.

Based on these results, the disposition of the nodes and links in the topology is a critical factor to consider when selecting an algorithm.

Further improvements were made when these algorithms were paired with continuous learning. By continuing the training process, the agents can better adapt to shifts and negative impact on network performance from changes in the topology was attenuated.

6.2 Future work

Some key elements that could optimize the techniques for scalable DRL algorithms are the state space composition and the usage of transfer learning. These factors are considered to be areas of improvement for applying trained agents to different topologies.

6.2.1 State space

The common approach is to use a static state space, although that restricts the DRL model and its adaptability. Ideally, the model would be dynamic, and evolve according to the topology characteristics. This issue if left to be explored in future work.

6.2.2 Transfer learning

Using TL with GNNs, which are a category of NNs specially suited for data represented in graph form, is a promising solution for scalable routing. Displaying data in graph form can help categorize new information by correlation. Each node, has an associated label and by using its neighboring relations, it is possible to estimate which label is associated with a new piece of data [13] [30].

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] Z. Xu et al. "Experience-driven Networking: A Deep Reinforcement Learning based Approach". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 1871–1879. DOI: 10.1109/INFOCOM.2018.8485853 (cit. on pp. 1, 13).

[3] M. Karakus and A. Durresi. "Quality of Service (QoS) in Software Defined Networking (SDN): A survey". In: *Journal of Network and Computer Applications* 80 (2016-12), pp. 200–218. DOI: 10.1016/j.jnca.2016.12.019 (cit. on p. 1).

[4] O. Guéant and I. Manziuk. *Deep reinforcement learning for market making in corporate bonds: beating the curse of dimensionality*. 2019. DOI: 10.48550/ARXIV.1910.13205. URL: https://arxiv.org/abs/1910.13205 (cit. on p. 1).

[5] A. S. Tanenbaum and D. Wetherall. *Computer Networks*. 5th ed. Boston: Prentice Hall, 2011. ISBN: 978-0-13-212695-3 (cit. on pp. 4, 5).

[6] H. Jiang. *Machine Learning Fundamentals: A Concise Introduction*. Cambridge University Press, 2021. DOI: 10.1017/9781108938051 (cit. on p. 5).

[7] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. 2nd ed. Adaptive Computation and Machine Learning. MIT Press, 2018. ISBN: 978-0-262-03940-6 (cit. on p. 5).

[8] S. V. Mahadevkar et al. "A Review on Machine Learning Styles in Computer Vision—Techniques and Future Directions". In: *IEEE Access* 10 (2022), pp. 107293–107329. DOI: 10.1109/ACCESS.2022.3209825 (cit. on pp. 5, 13, 14).

[9] L. Bougrain. "Practical Introduction to Artificial Neural Networks". In: *IFAC Proceedings Volumes* 37.15 (2004). 11th IFAC Symposium on Automation in Mining, Mineral and Metal Processing (MMM'04), Nancy, France, September 8-10, 2004, pp. 347–352. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)31048-0. URL: https://www.sciencedirect.com/science/article/pii/S1474667017310480 (cit. on p. 6).

[10] Z. M. Fadlullah et al. "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems". In: *IEEE Communications Surveys Tutorials* 19.4 (2017), pp. 2432–2455. DOI: 10.1109/COMST.2017.2707140 (cit. on p. 6).

[11] *What is machine learning?* 2023. URL: https://www.ibm.com/topics/machine-learning (cit. on p. 7).

[12] S. Z. Hao Dong Zihan Ding. *Deep Reinforcement Learning: Fundamentals, Research, and Applications*. Ed. by S. Z. Hao Dong Zihan Ding. http://www.deepreinforcementlearningbook.org. Springer Nature, 2020 (cit. on pp. 7–12).

[13] X.-D. Zhang. *A Matrix Algebra Approach to Artificial Intelligence*. 1st ed. Springer Nature, 2020. DOI: https://doi.org/10.1007/978-981-15-2770-8 (cit. on pp. 8–10, 59).

[14] T. Dong et al. "Generative Adversarial Network-Based Transfer Reinforcement Learning for Routing With Prior Knowledge". In: *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT* 18 (2 2021), p. 1673. DOI: 10.1109/TNSM.2021.3077249. URL: https://www.ieee.org/publications/rights/index.html (cit. on p. 8).

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018 (cit. on p. 9).

[16] T. Li et al. "Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 24 (2021), pp. 1240–1279. URL: https://api.semanticscholar.org/CorpusID:239885710 (cit. on p. 9).

[17] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG] (cit. on p. 9).

[18] K. Arulkumaran et al. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/MSP.2017.2743240 (cit. on p. 10).

[19] T. P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG] (cit. on p. 12).

[20] W. Menghao et al. "The Actor-Dueling-Critic Method for Reinforcement Learning". In: *Sensors* 19 (2019-03), p. 1547. DOI: 10.3390/s19071547 (cit. on p. 12).

[21] P. Sun et al. "A Scalable Deep Reinforcement Learning Approach for Traffic Engineering Based on Link Control". In: *IEEE Communications Letters* 25 (1 2020), pp. 171–175. ISSN: 1089-7798. DOI: 10.1109/lcomm.2020.3022064 (cit. on pp. 15, 16).

[22] C. Yu et al. "DROM: Optimizing the Routing in Software-Defined Networks with Deep Reinforcement Learning". In: *IEEE Access* 6 (2018). The related wprk is not well justified. especially with regards to references12 and 11., pp. 64533–64539. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2877686 (cit. on p. 15).

[23] P. Sun et al. "Enabling Scalable Routing in Software-Defined Networks With Deep Reinforcement Learning on Critical Nodes and also with the National Digital Switching System Engineering and". In: *IEEE/ACM TRANSACTIONS ON NETWORKING* 30 (2 2022), p. 629. DOI: 10.1109/TNET.2021.3126933. URL: https://www.ieee.org/publications/rights/index.html (cit. on p. 16).

[24] P. Almasan et al. "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case". In: *Computer Communications* 196 (2022-12), pp. 184–194. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2022.09.029. URL: http://dx.doi.org/10.1016/j.comcom.2022.09.029 (cit. on p. 16).

[25] N. Geng et al. "A Multi-agent Reinforcement Learning Perspective on Distributed Traffic Engineering". In: *Proceedings - International Conference on Network Protocols, ICNP* 2020-Octob (2020). ISSN: 10921648. DOI: 10.1109/ICNP49622.2020.9259413 (cit. on pp. 17, 20, 23).

[26] X. Zhao, C. Wu, and F. Le. "Improving Inter-domain Routing through Multi-agent Reinforcement Learning". In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2020), pp. 1129–1134. DOI: 10.1109/INFOCOMWKSHPS50562.2020.9162984 (cit. on pp. 18, 20).

[27] P. Pinyoanuntapong, M. Lee, and P. Wang. "Delay-Optimal Traffic Engineering through Multi-agent Reinforcement Learning". In: *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019* (2019), pp. 435–442. DOI: 10.1109/INFCOMW.2019.8845154 (cit. on pp. 18, 20).

[28] R. Lowe et al. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. 2020. arXiv: 1706.02275 [cs.LG] (cit. on p. 19).

[29] Z. Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: 1511.06581 [cs.LG] (cit. on p. 28).

[30] S. Raschka, Y. (Liu, and V. Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Birmingham, UK: Packt Publishing, 2022. ISBN: 978-1801819312 (cit. on p. 59).